A Load Balancing Algorithm for Distributed Computing Systems

by

Mohammed Homoud Melhi

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

January, 1990

A LOAD BALANCING ALGORITHM FOR DISTRIBUTED COMPUTING SYSTEMS

BY

MOHAMMED HOMOUD MELHI

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In

COMPUTER SCIENCE

JANUARY 1990

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by ***MOHAMMED HOMOUD MOHAMMED MELHI*** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of ***MASTER OF SCIENCE*** in ***COMPUTER SCIENCE AND ENGINEERING.***

THESIS COMMITEE

M Bozyigit
Dr. M. Bozyigit (Chairman)

S. Ghanta
Dr. S. Ghanta (Member)

Ü. Kalaycioğlu
Dr. U. Kalaycioglu (Member)

Department Chairman

Abdel
F| *Dean, College of Graduate Studies*

College of Graduate Studies
Office of the Dean

ACKNOWLEDGEMENT

Acknowledgement is due to King Fahd University of Petroleum and Minerals for support of this research.

I would like to express my thanks to Dr. Muslim Bozyigit, my major advisor, for providing me active guidance, help and encouragements. I would also wish to thank other committee members, Drs. Subbarao Ghanta and Unsal Kalaycioglu for their help and valuable suggestions. My gratitude also goes to the Chairman of ICS Dept. for his support in using the departmental facilities.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

خـــلاصـــة الـرســالـة

اسم الطالب : محمد حمود محمد ملهى.

عنوان الدراسة : موازنة التحميل فى أنظمة الكمبيوتر الموزعة .
التخصص : علوم وهندسة الحاسب الآلى.
تاريخ الدرجة : يناير ١٩٩٠م

فى هذه الرسالة تم إقتراح نموذج جديد لموازنة التحميل فى انظمة الكمبيوتر الموزعة ، ويعتمد هذا النموذج على مفهوم المرحلة الزمنية فى التطبيقات الموزعة حيث يفترض أن يمر التطبيق الموزع بعدة مراحل زمنية، فى كل مرحلة يمكن تنفيذ عدة أجزاء من التطبيق على التوازى (أى فى نفس الوقت) وتتصل هذه الاجزاء ببعضها على التوازى أيضا.

لقد تم ابتكار برنامج يقوم بتوزيع أجزاء التطبيق على وحدات نظام الكمبيوتر الموزع بطريقة فعالة بحيث ينفذ هذا التطبيق بواسطة النظام فى وقت أسرع.

يعتمد هذا البرنامج على تقسيم أجزاء التطبيق إلى مجموعات يلى ذلك التعيين المبدئى ثم التبادل المزدوج الفعال لمواضع هذه المجموعات. وقد تم اختبار هذا البرنامج على تمثيل بيانى لعدد من شبكات أنظمة الكمبيوتر الموزعة والتطبيقات تم الحصول عليها عشوائيا . وبعد مقارنة نتائج البرنامج مع النتائج المثالية تبين أن البرنامج ناجح إلى حد كبير فى الوصول إلى حلول قريبة من المثالية. إضافة إلى ذلك تم تعزيز البرنامج بطريقتين . الطريقة الأولى تقلل من الوقت اللازم لتنفيذ البرنامج والطريقة الثانية تعطى نتائج أفضل بواسطة تغيير مواضع أجزاء التطبيق.

درجة الماجستير فى العلوم
جامعة الملك فهد للبترول والمعادن
الظهران - المملكة العربية السعودية
ينايـــر - ١٩٩٠م.

ABSTRACT

Mohammed Homoud Mohammed Melhi

A LOAD BALANCING ALGORITHM FOR DISTRIBUTED COMPUTING SYSTEMS

Major Field : Computer Science & Engineering

January 1990

In this thesis we propose a new model of Load Balancing for distributed computing systems. The model is based on a phase concept of a distributed application which is defined as "a contiguous period of time during which a number of application modules execute in parallel and communicate in parallel". The algorithm developed consists of graph reduction, initial assignment of modules and pairwise exchange techniques. It has been tested on a number of distributed applications and distributed systems generated randomly. The test results showed that the algorithm is successful in achieving suboptimal solutions compared to the optimal solutions. In addition, two enhancements to the algorithm have been incorporated. The first enhancement reduces complexity of the algorithm. The second enhancement improves objective function by means of module reassignment.

MASTER OF SCIENCE

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS
DHAHRAN, SAUDI ARABIA

January 1990

CHAPTER 1

INTRODUCTION

A recent trend in computer systems is to distribute computation among a number of (micro)-processors. Microprocessors, compared to mainframes and minicomputers have become cheap and reasonably powerful; and this motivates researchers to dig into the area of using microprocessors as building blocks for multiprocessing systems. Basically, there are two schemes for building such systems: tightly coupled systems and loosely coupled systems. In tightly coupled systems, generally, processors communicate through a shared memory or a shared bus. The system shown in Figure 1.1 is a typical example of tightly coupled systems. On the other hand, each processor in loosely coupled systems has its own memory and communicate with other processors through various types of communication lines such as telephone lines , high speed buses and serial communication links as depicted in Figure 1.2. Those communication lines can be half-duplex or full-duplex. We refer to loosely coupled multiprocessor systems as distributed computer systems(DCS).

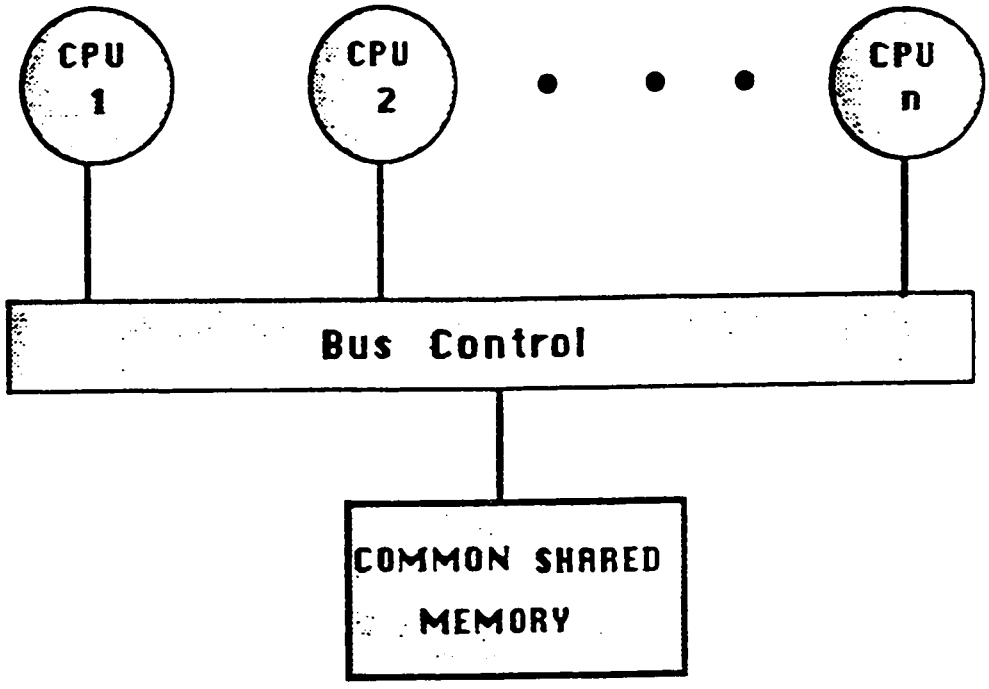


Figure 1.1 : A Typical Tightly Coupled System.

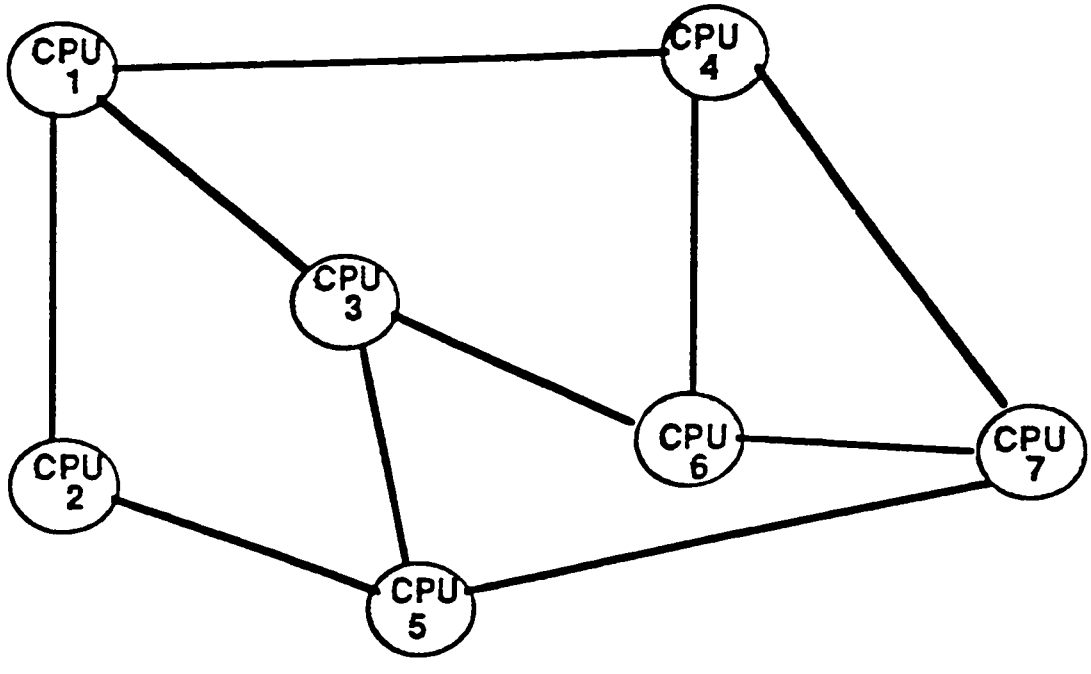


Figure 1.2 : A Loosely Coupled System (Each CPU has its own local memory) .

Many application programs can be divided into a number of cooperating programs that may run concurrently on different processors resulting in reduction of the total execution time of the application. This objective attracts people to use DCS. Moreover, real-time systems, used for example in nuclear power plants and process control applications, are one class of the systems that get benefit from using DCS because they can have severe timing constraints and because applications are themselves distributed. DCS have a number of advantages ,other than computation speed-up. The most significant ones are : resource sharing, reliability, and availability [TAN85,PET85,STAN84]. Resource sharing is due to the availability of communication networks that allow users residing in wide areas to share hardware and software resources. Computation speed-up and availability are due to the availability of a number of processors. Data redundancy and fault tolerant control make DCS highly reliable; even if part of a DCS fails, the system can continue functioning.

On the other hand, there are some basic problems faced in designing DCS such as load balancing, communication protocols, reliability, synchronization, coordination (deadlock and consistency) and reconfigureability. To narrow our study domain, we restrict our attention to load balancing problem only.

Load Balancing

Load balancing is a policy for assigning a number of (dependent or independent) programs to processors in a DCS such that a number of goals(objectives) are satisfied. Some examples of such goals are throughput maximization, even distribution of work load among processors, and minimizing communication cost between the communicating programs. These goals are inter-related, and usually researchers focus on one or more goals.

In the past decade, the load balancing problem has received a lot of attention by computer researchers. There are many different approaches for this problem that can be grouped into two classes, namely static and dynamic approaches. In all formulations of the problem, the problem of obtaining optimal assignment of application modules to processors is found to be NP-complete [BOKH81], except for systems that have a number of processors less than four, yet with many restrictions. Bokhari proved that the assignment of modules to processors, which is equivalent to the mapping problem, to be not computationally tractable by showing that the mapping problem, in its most general form, is equivalent to the graph homomorphism problem. The graph homomorphism problem is one of the well known intractable combinatorial

problems. He assumes that the number of modules is equal to the number of processors which is a special case of the mapping problem and still he found the problem to be intractable. For this reason research has focussed on development of heuristic algorithms to find sub-optimal solutions at least in polynomial times. For the sake of providing enough background on the problem, Chapter 2 will discuss some of the basic strategies for the assignment problem. Chapters 3 and 4 contain the proposed load balancing algorithm and its implementation respectively. Analysis of the algorithm is shown in Chapter 5. Finally, conclusions and directions for future research in this area is provided.

For the rest of discussion, the terms modules, processes, and tasks are used interchangeably. Also, processor and CPU are considered to be the same.

CHAPTER 2

LOAD BALANCING : A REVIEW

2.1 Static Approaches :

Methods that assume fixed execution and communication requirements during the run time of the application are referred to as static load balancing approaches. Many load balancing approaches of this type have been proposed [STON77,BOKH81,LO88]. These approaches are guided by some objectives or constraints which may be different from approach to approach. Formally, we can express a constraint or the set of constraints in the form of an objective function(OF). In some cases, these OF's need to be minimized and in others they need maximization. Some examples of the OF's to be minimized are : communication overhead between processors [LEE87,EFE82], completion time of the application program [AGR88],both communication and execution times [STON77,STON78]. Examples of the OF's needing maximization are : cardinality [BOKH81], and system throughput [BARA85]. The cardinality of mapping a problem graph onto a system graph is defined as the number of edges in the problem graph

that fall on the edges of the system graph. The problem graph is a graph that describes the communication requirements between modules, while the system graph describes the interconnection of processors in the DCS. There is a general agreement about including inter-process communication(IPC) and execution cost together in the OFs. Moreover, all objective functions mentioned above are to some extent in conflict, and many researchers try to evaluate different trade-offs. The above mentioned objectives can be used in dynamic load balancing as we shall see in section 2.2. In dynamic load balancing, execution and communication requirements of application and/or processors load can be changed dynamically from time to time. The following sections will briefly describe five static load balancing approaches.

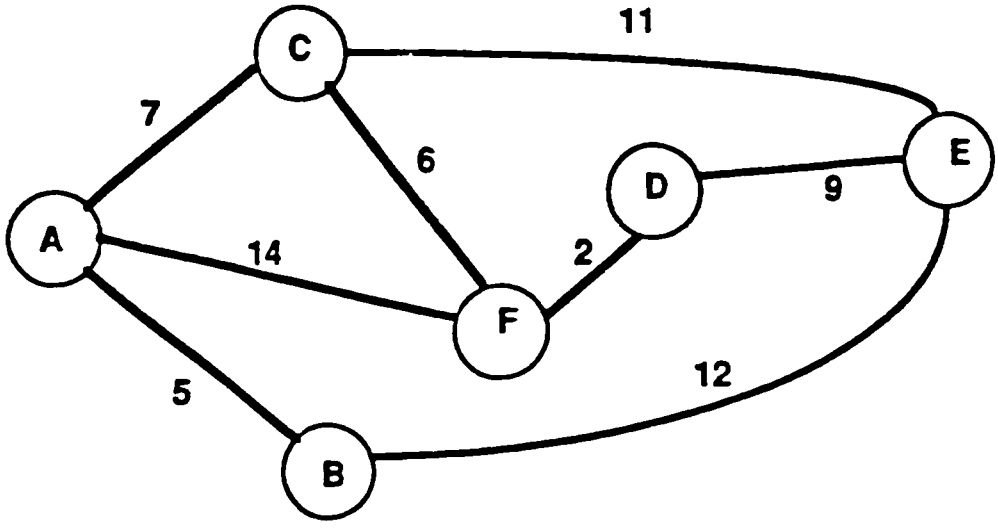
2.1.1 Stones Approach

This approach is based on graph theoretical approaches [STON77,STON78]. The number of processors is limited to two and the distributed application is modeled by a graph. Nodes in this graph correspond to the application modules and presence of an edge corresponds to the existence of communication between the modules linked by that edge. The edge weights represent the communication cost between

modules provided that both modules are not assigned to the same processor. The execution cost is represented in the form of an execution matrix. Figure 2.1 shows an example of an application graph and an execution matrix.

A module M_i with execution cost equal to infinity in the execution matrix indicates that M_i cannot be executed on that processor. The program graph is modified by adding two special nodes (P_1 and P_2) corresponding to the two processors in the system. Also, edges are added connecting each processor to every module. The weight of the new edges are such that the weight of an edge from a module M to P_1 is equal to execution cost of M on P_2 and the weight of the edge from M to P_2 is the execution cost of M on P_1. The modified program graph in Figure 2.1 is shown in Figure 2.2. Stone utilized a Maximum Flow/Minimum Cut algorithm to find the assignment that minimizes both communication and execution costs. He does that by selecting a cutset that correspond to the optimum assignment. A cutset is a collection of edges such that [STON78]:

- when removed, node P_1 is disconnected from P_2.
- no proper subset of a cutset is a cutset.



(a) An Appliction Graph

Module	Execution cost on P1	Execution cost on P2
A	4	12
B	6	7
C	5	infinity
D	2	3
E	infinity	4
F	1	2

(b) Execution Cost of modules

Figure 2.1 : Requirements of an Application

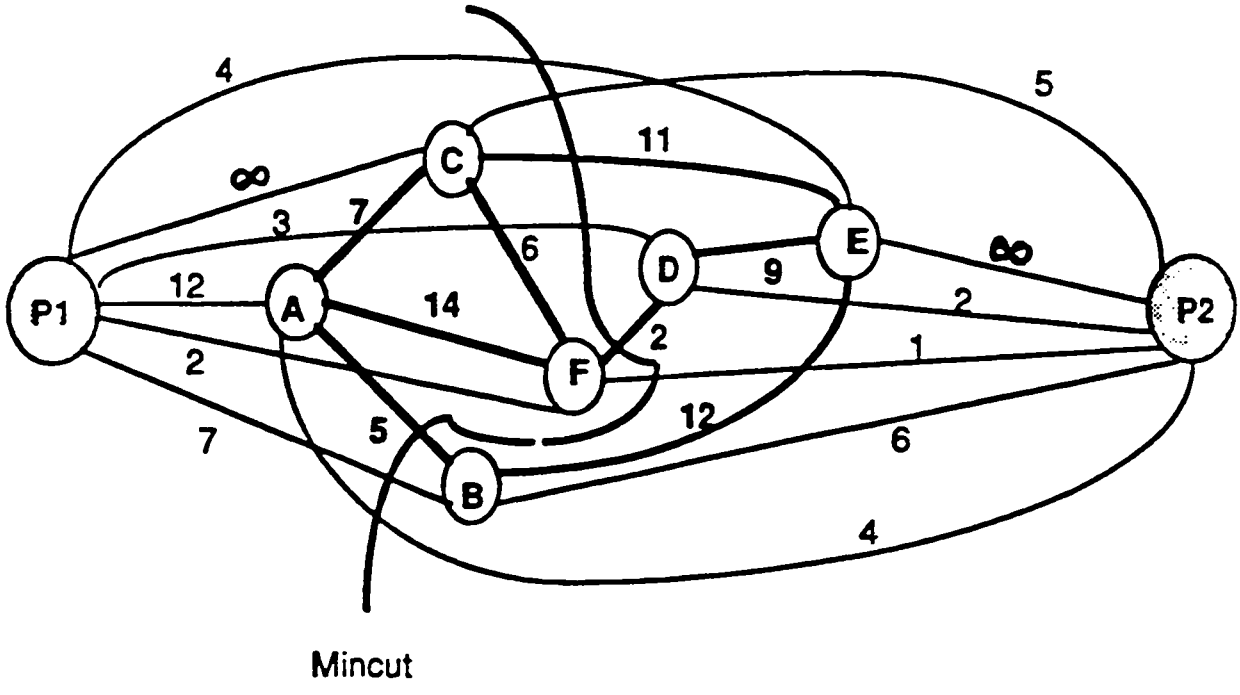


Figure 2.2 : Application-Processor Graph.

The weight of a cutset is the sum of weights it cuts. The dark line in Figure 2.2 shows the cutset that correspond to the optimum assignment. It is proven that the correspondence between cutsets and module assignments is one-to-one, and the minimum cutset will give the optimum module assignment for dual processors. Stone's model assumed that only one module is active at a time; so no concurrency is allowed in this approach. A good advantage of this approach is that it is optimal for dual processors. However, it is not optimal for more than two processors , because many problems arise when we use this method for a system with three or more processors, simply the Maximum Flow/Minimum cut algorithm is applicable to a source (P_1) and a sink (P_2). For a DCS with 3 processors, the application graph is modified by connecting edges from every module to all processor nodes as in the dual processor case[STON78]. However, the weights of those edges are computed differently. Let the execution cost of a module M on processor P_i be t_i, for i = 1,2,3. Then, the weight of the edge (P_1,M) is computed as $(\frac{t_2+t_3-t_1}{2})$. Similarly, the weight of the edge (P_2,M) is computed as $(\frac{t_1+t_3-t_2}{2})$ and the weight of edge (P_3,M) is equal to

$(\frac{t_1+t_2-t_3}{2})$. After modifying the graph, the Maximum Flow / Minimum Cut algorithm is used for every pair of processors (one of them is designated as the source and the other one as the sink) to find a minimum tri-cutset which correspond the near optimal assignment.

2.1.2 Lo's Approach

This method is a heuristic graph theoretical approach and it extends the Stone's approach to any number of processors [LO88]. Parallelism is also considered but as a secondary goal by introducing interference cost. The module processor graph construction is similar to stone's. However, the number of processors can be more than two , and the weights of edges connecting processor to modules is computed by the following formula :

$$W_{iq} = \frac{1}{n-1} \sum_{r \neq q} X_{ir} - \frac{n-2}{n-1} X_{iq}$$

Where W_{iq} is the weight of the edge from P_q task t_i, X_{ir} to task t_i, X_{ir} is the execution cost of t_i on P_r, and n is the total number of processors. See Figure 2.3 for an example of such a graph.

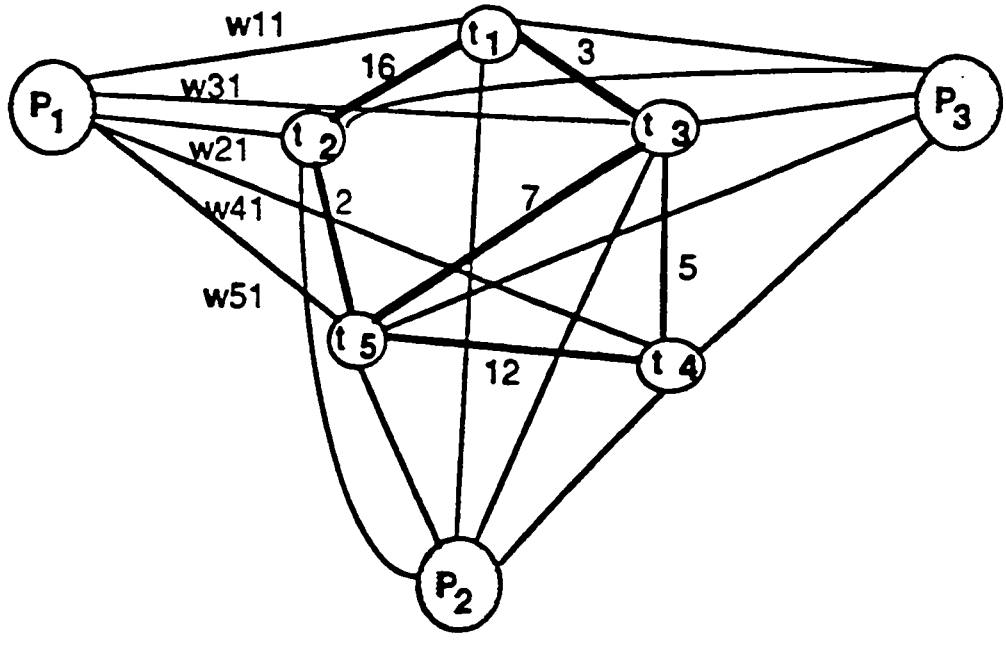


Figure 2.3: Application Graph with Three Processors.

For every P_i, this graph is converted into a reduced graph that contain two processor nodes P_i and $\overline{P}_i$ instead of n nodes as shown in Figure 2.4 for P_2. Node $\overline{P}_i$ is a hypothetical node representing all nodes other than P_i. The weight of the edge connecting $\overline{P}_i$ to a task t_j is the sum of all weights of the edges from tj to all processors in the original graph except P_i. The Maximum Flow/Minimum Cut algorithm is utilized to obtain the set of tasks that have to be allocated to P_i and we drop the set allocated to $\overline{P}_i$. The same thing is done for all processors in the DCS. If no tasks remain without assignment, the algorithm terminates and assignment is optimal. However, if a number of modules remain without assignment then two procedures(called Lump and Greedy) are invoked to assign the modules to one of the processors or distribute them onto more than one processor, respectively. The time complexity of this algorithm is $O(nk^2e \log k)$ where e is the number of edges in the original graph, n is the number of processors, and k is the number of tasks.

The above extension of Stone's model has a serious deficiency: It does not include concurrency as an objective

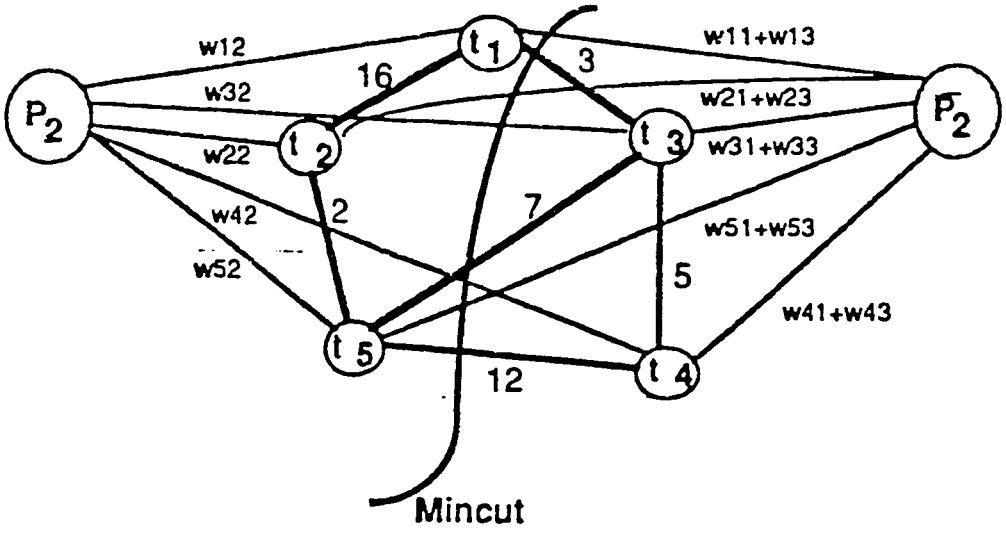


Figure 2.4: Reduced Graph of Figure 2.3 for processor P2.

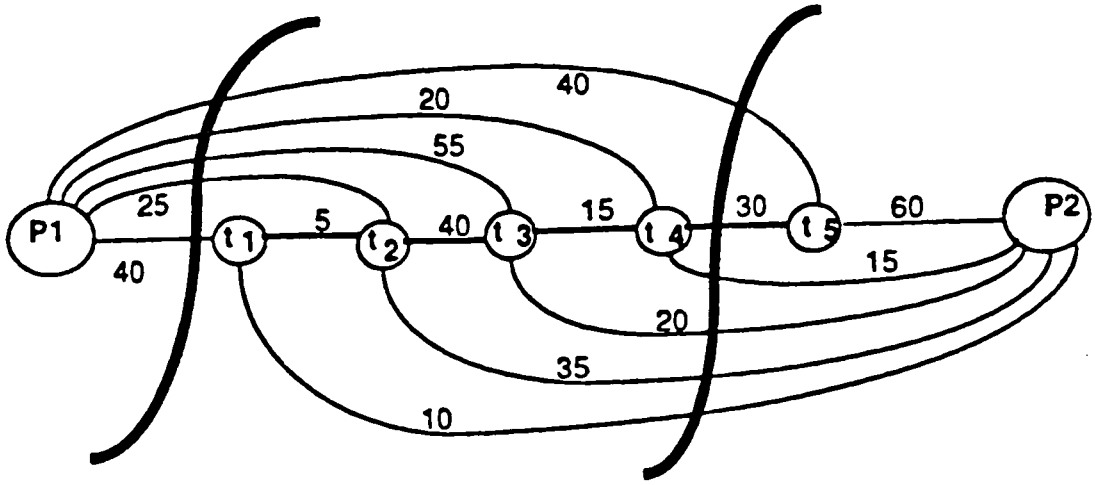
function to be achieved. Since execution and communication costs are the only costs that have been minimized, the algorithm yields assignments that utilize portion of the processing power(i.e fewer processors are utilized). The cause of such deficiency is that modules assigned to one processor can't run in parallel, and they should be multiprogrammed. The tendency of assigning two modules to the same processor is increased by increasing communication cost. Let us assume that we have an application that consists of five tasks ($t_1,\ldots,t_5$) to be assigned to a dual processor system. The execution and communication costs of the tasks is shown in Figure 2.5(a). The two cutsets shown in the figure have equal weights, and both are minimum and they can be produced by the algorithm. However, the cutset on the right allows more parallelism.

To overcome the deficiency presented above slightly, the network flow algorithm is extended by considering another factor called interference cost. Interference costs are associated with any two modules when both are assigned to the same processor as a penalty for sharing this resource(i.e. because of decreasing degree of parallelism). Two CPU bounded tasks are given interference cost greater that two I/O bounded modules. This interference cost is not

Execution costs		
	P1	P2
t1	10	40
t2	35	25
t3	20	55
t4	15	20
t5	60	15

Communication costs					
	t1	t2	t3	t4	t5
t1	0	5	0	0	0
t2	0	0	40	0	0
t3	0	0	0	15	0
t4	0	0	0	0	30
t5	0	0	0	0	0

(a)



(b)

Figure 2.5 : Two Mincuts with the Same Cost (one of them allows more concurrency).

included in the total cost if the two modules are assigned to two different processors. The extension of the network flow algorithm is done by replacing the weights of the edges connecting processor q to task t_i with the following weight:

$$w_{iq} = \frac{1}{n-1} \sum_{r \neq q} X_{ir} + \frac{1}{2(n-1)} \sum_{1 \leq l \leq k} I_{il}$$

and by replacing the weights of edges connecting task t_i to task t_j with the following weights

$$C'_{ij} = c_{ij} - I_{ij}$$

where I_{ij} is the interference costs between t_i and t_j. The same algorithm is applied to the resulting graph. Simulation results show that considering interference costs yield better assignments with greater parallelism.

2.1.3 Efe's Approach :

In this approach [EFE82], The DCS is assumed to be homogeneous and fully connected. The method consists of two phases and is based on two algorithms: Module Clustering Algorithm(MCA) and Module Reassignment Algorithm(MRA). In the first phase, the MCA is used to partition the set of

modules into k subsets considering ,only, the communication cost. A search is made to find the k that maximizes a load balancing among all k = 2,....,n, where n is the number of processors. A criterion is developed to check the goodness of the load in all processors: A system is considered to be balanced if the queue lengths (i. e. the time needed to execute modules assigned to a processor) of all processors in the DCS fall within a given tolerance that is supplied by the user. So, the queue length (l_i) of P_i is the sum of the execution costs of modules assigned to P_i. We terminate the algorithm if this criterion is satisfied; otherwise we perform phase 2.

In the second phase, overloaded and underloaded processors are identified, and we call the MRA to assign modules from the overloaded processors to the underloaded processors such that the criterion is satisfied.

2.1.4 Bokhari's Static load assignment Approach :

Bokhari proposed a number of strategies for the assignment problem [BOKH81,BOKH88]. In one of these methods, he reduces the problem into the mapping problem [BOKH81].

The mathematical formulation of this method is as follows:

Let $G_p = \langle V_p, E_p \rangle$ be a representation of the application to be mapped onto an array of processors. The vertices of V_p correspond to the set of modules, and each edge (V_{pi}, V_{pj}) in E_p denotes that module V_{pi} communicates with module V_{pj}. Edges are not given weights. Let $G_a = \langle V_a, E_a \rangle$ be the graph of the array processor, where V_a is the set of processors and the edges E_a represent the processor interconnections. Now, the problem is to map $V_p \rightarrow V_a$. He considered the cases where $|V_p| <= |V_a|$. When $|V_p| < |V_a|$, dummy vertices may be added to V_p, so that $|V_p| = |V_a|$. The quality of a mapping is determined by the number of problem edges that fall on system edges after the mapping.

Pairwise exchange is used to improve the quality of the mapping. The pairwise exchange is terminated when there is no pair that can improve the mapping any more. Empirical results show that the number of exchanges is not too large.

Many disadvantages are inherent with this approach. First, the edges that do not fall onto system edges are ignored, and in some cases these edges may directly affect

the run time of the application. In addition, communication costs are not considered at all. Finally, this approach can't be used for $|V_p|>|V_a|$.

Another approach, also proposed by Bokhari, uses a different objective function: The sum of products of the weights of the problem edges and the distance (number of edges) between the corresponding nodes in the system graph. This objective function ignores the effect of the bottleneck system links that may result from mapping more than one problem edge onto one of links.

2.1.5 Lee's Mapping strategy :

Most application assignment approaches assume fixed communication cost between two modules. In other words, communication cost is not sensitive to module location. But this assumption is not right because the cost of communication m_1 and m_2 according to the assignment of Figure 2.6(b) should be greater than that of the assignment in Figure 2.6(a) because (in Figure 2.6(b)) the messages exchanged between m_1 and m_2 are buffered by P_2 and this will incur more time delay.

Lee and Aggarwal [LEE87] proposed a mapping strategy that

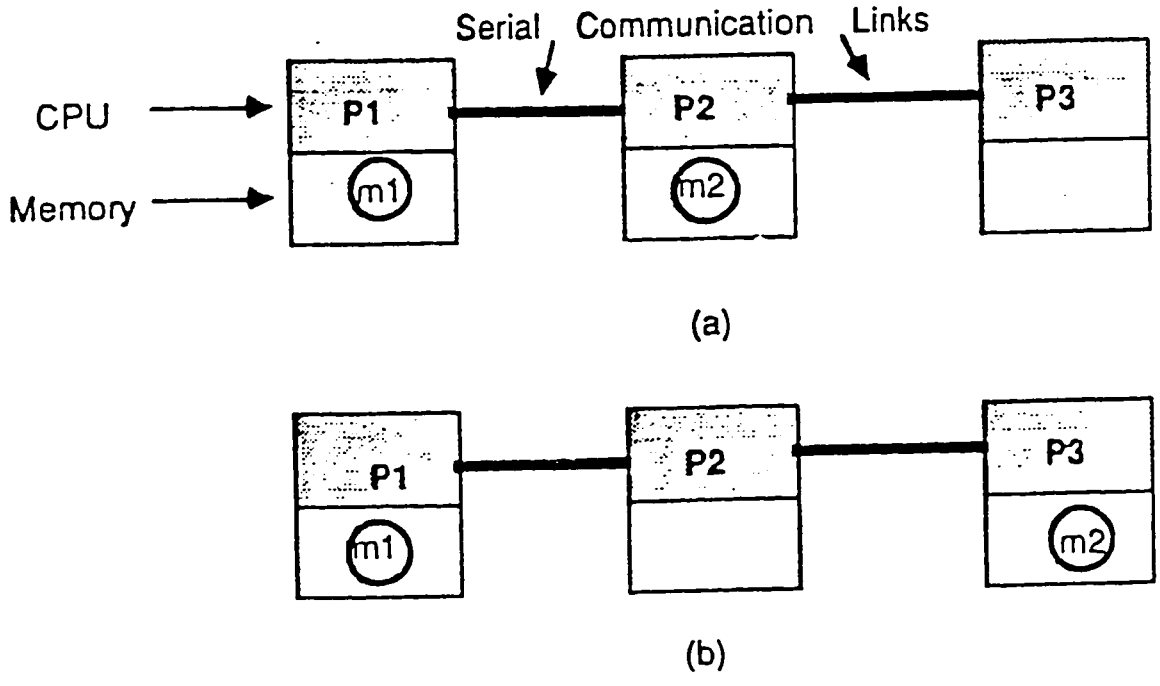
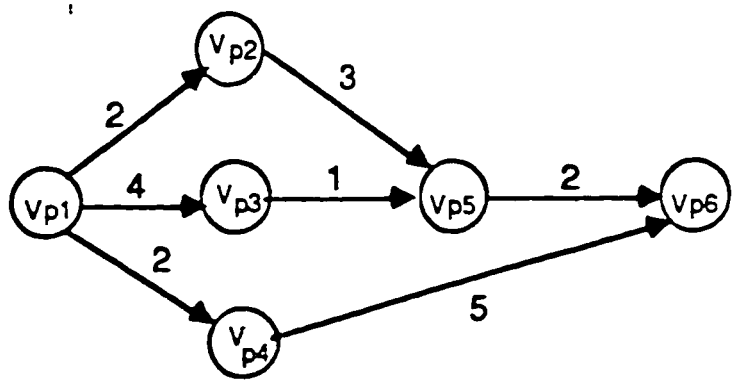
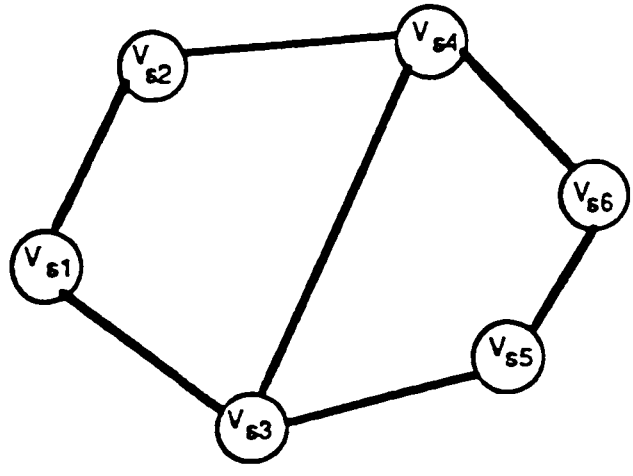


Figure 2.6: Two Possible Locations for two modules(m1&m2).

takes this point into consideration. The strategy is used for distributed parallel processing applications and it maps a problem graph with n processes onto a system graph of n processors. Examples of both graphs are shown in Figure 2.7. Note that saying "mapping processes onto processors" is equivalent to saying "mapping problem edges onto system edges." Four objective functions (OF_1,OF_2,OF_3,OF_4) are described [LEE87] to quantify correctly the real overhead incurred by module communication. OF_1 is suitable for sequential processing, OF_2 is suitable for pure parallel processing, OF_3 is suitable for application that require sequential and parallel processing (because of precedence conditions), and OF_4 is suitable for pipelined processing. Execution costs are not given any attention because the number of processes equals the number of processors and because processors are assumed to have the same capabilities. Before evaluating an OF we need to sort edges (according to the time of activating those edges) and then we need to evaluate the communication overheads of all communication cost (C_{ij}) corresponding to communication edges between process m_i and m_j. The communication overhead can be interpreted as the time needed for sending messages from a



(a) A Problem Graph



(b) A System Graph

Figure 2.7: A Distributed Parallel Processing Model.

source to a destination. The algorithm that evaluate the C_{ij}'s is outlined below. This algorithm produces a matrix C that contains all C_{ij}'s. Note that the C matrix and the OF employed are calculated for a given mapping.

PROCEDURE Communication Cost Calculation;

- Sort the set of problem edges (E_p) into subsets (E_{pq}) according to time of edge activation (subset E_{pq} contains edges that are activated at time q).
- For each E_{pq} do

begin

.. For each problem edge $e_{pij} \in E_{pq}$ find the nominal distance of the corresponding system edge(e_{skl}).

.. For each e_{pij} whose corresponding system edge e_{skl} has a nominal distance > 1 do

... Decompose the path e_{skl} into a sequence of links(or nodes) depending on the topology of the system and routing rules employed.

.. For each e_{pqj} do

... Assign the packets to processor v_{sk} such that the number of packets is equal to the weight of e_{pij}.

... Associate the following data structures with each packet:

SOURCE = v_{sk}

DESTINATION = v_{sl}

STEP = 0,

STEP is used to indicate the number of steps the packet has spent in reaching the current node.

.. Repeat the following

... For all packets set STEP = STEP + 1 ;

... For each system node v_{sl} Do

.... For one or more of the links connected to v_{sl} (depending on the

```
                    communication protocols)  Do
                    .... Move a packet(or a number of
                         packets if it is allowed by
                         the communication protocols
                         of the distributed system)
                         to the next node. The next
                         node is determined by the
                         routing rules of the DCS.
          ... If a packet reaches its destination then
              remove the packet if it is the last
              packet from its source then
                   C_ij = STEP of the last packet from
                   v_sk to v_sl.
               Else
                   Add it to the queue of the current
                   node.
        Until all packets reach their destinations.
End.
```

After evaluating C_{ij}'s the OF that is employed in the system (i.e OF_1, OF_2, OF_3, or OF_4 according to the required type of processing, i.e sequential, parallel, hybrid, or piplined, respectively) is evaluated.

The definitions of these OF's are:

$$OF_1 \triangleq \sum_{ij} (C_{ij})$$

$$OF_2 \triangleq \max(C_{ij}) \text{ Over all } C_{ij}\text{'s}$$

$$OF_3 \triangleq \sum_{q} (\max_{j} (C_{ij})) \text{ Over all subsets } E_{pq}\text{'s}$$

$OF_4 \triangleq \max_q \ (\max_j(C_{ij}))$ Over all subsets E_{pq}'s.

The proposed mapping strategy consists of two procedures : Initial Assignments procedure and Pairwise Exchange procedure.The goal of the former procedure is to try to obtain a mapping with as small OF as possible. If the mapping achieved is not satisfactory, the mapping is optimized by calling the latter procedure. Both of these procedures are presented below:

```
PROCEDURE Initial Assignments;
begin
      . Search for a problem node v_pi with the largest
        communication intensity and assign this node to a
        system node v_sj such that the degree of
        v_pi (d(v_pi)) is as close to d(v_sj). (*communication
        intensity of v_pi is the sum of all weights of edges
        connected to this node*)
      . Remove V_pi from the set of problem nodes V_p.
      . While |V_p| > 0 do
          begin
            .. Find a problem node v_pk with the largest
               communication intensity that is adjacent to
               the problem nodes already assigned.
            .. Assign v_pk to a system node v_sl such that the
               OF employed is optimized .
            .. Remove v_pk from v_p.
    end;
end.

PROCEDURE Pairwise Exchange;
begin
            .. Evaluate the OF of the mapping obtained in the
               initial assignment.
```

```
.. Repeat
     ... Find a candidate problem node v_pi for
         exchange. The candidate can be the node
         that has the largest communication
         intensity.
     ... For all nodes v_pj do
          .... Exchange v_pi with v_pj temporarily.
          .... Evaluate the OF of the resulting
               mapping.
     ... Among all nodes exchanged select v_pl that
         give the smallest OF (call it OF_new).
     ... If OF_new is smaller than the older OF
         then
               Make the exchange (v_pi <->
         v_pl) permanent.
          Else
               Output the current mapping
               Exit
   endif
end.
```

The complexity of the initial assignment procedure is $O(N^2)$ where $N = |Vp|$, and the complexity of the pairwise exchange procedure is $O(N_i N^2 D_M \alpha_M)$ where N_i is the number of iterations in the pair wise exchange, D_M is the diameter of the system graph (i.e the maximum distance between nodes), and α_M is the maximum edge weight in a problem graph. Because of using the initial assignment the number of pairwise exchanges is very small as it is confirmed by simulation results [LEE87].

2.2 Dynamic Approaches :

There are two basic approaches for balancing the load of processors dynamically in DCS, either by exchanging load control messages between processors or by static determination of which/where modules should be relocated at program compilation(i.e. prior to application initiation). The following subsections give approaches of this type in the literature.

2.2.1 Load Exchange Approaches :

Some of the policies of this type can be found in [TAN85,BARA85,LIN87]. The basic idea behind these approaches is to use the load value of all processors in reducing the variance between loads dynamically. We will talk briefly about one of these approaches.

In [BARA85], the policy is summarized as follows . Each processor holds a load vector(L) of size l, and l is very small compared to n, where n is the number of processors). The objective of taking l<<n reduces load exchange messages. The first component of L holds the load value of the local processor, and the remaining components hold load value of an arbitrary subset of Vp, where Vp is the set of processors. Each processor executes continuously an

algorithm for calculating its local load during an interval(t). The load is evaluated as the ratio between the number of quantum requests and the number that could be serviced during t. Two other algorithms are executed by every processor namely, load exchange and process migration algorithm. The former algorithm is used for continuous exchange of portions of the load vector with a randomly selected processor. The latter algorithm is responsible for migrating process from the local processor(if it is overloaded) to another lightly loaded processor.

The main problems with the load exchange approaches are making processor busy in evaluating its load and sending load messages continuously, wasting the communication bandwidth, and wrong selection of processes for migration (e.g migration of processes that will terminate soon).

2.2.2 Program Compilation based Dynamic Load Balancing :

An algorithm of this type has been proposed by Bokhari [BOKH79]. He considered the dual process case only, as Stone did. The algorithm is based on the concept of phase of a modular program. The phase is defined as a period of time during which only one module executes. Unfortunately, concurrency is not considered in this approach. The mathematical model of this approach is the same as Stone's model except that each module node is replicated as many as the number of phases and an edge connects each module in phase i with the same module in phase i+1 with weight equals to the relocation cost of this module. The resulting graph is called dynamic assignment graph. Figure 2.8 shows an example of a dynamic assignment graph with three modules (A,B,C) and five phases. The Maximum Flow/Minimum Cut algorithm is used to find the optimal assignment. The dark line in Figure 2.8 shows the minimum cut that correspond to the optimal assignment. This cut tells, for example, that module B should be relocated from P_1 to P_2 at phase two.

This approach is not suitable for more than two processors because relocation cost increases as the path between two processors increases. Also, concurrency is not considered. In addition, the same disadvantages of Stone's

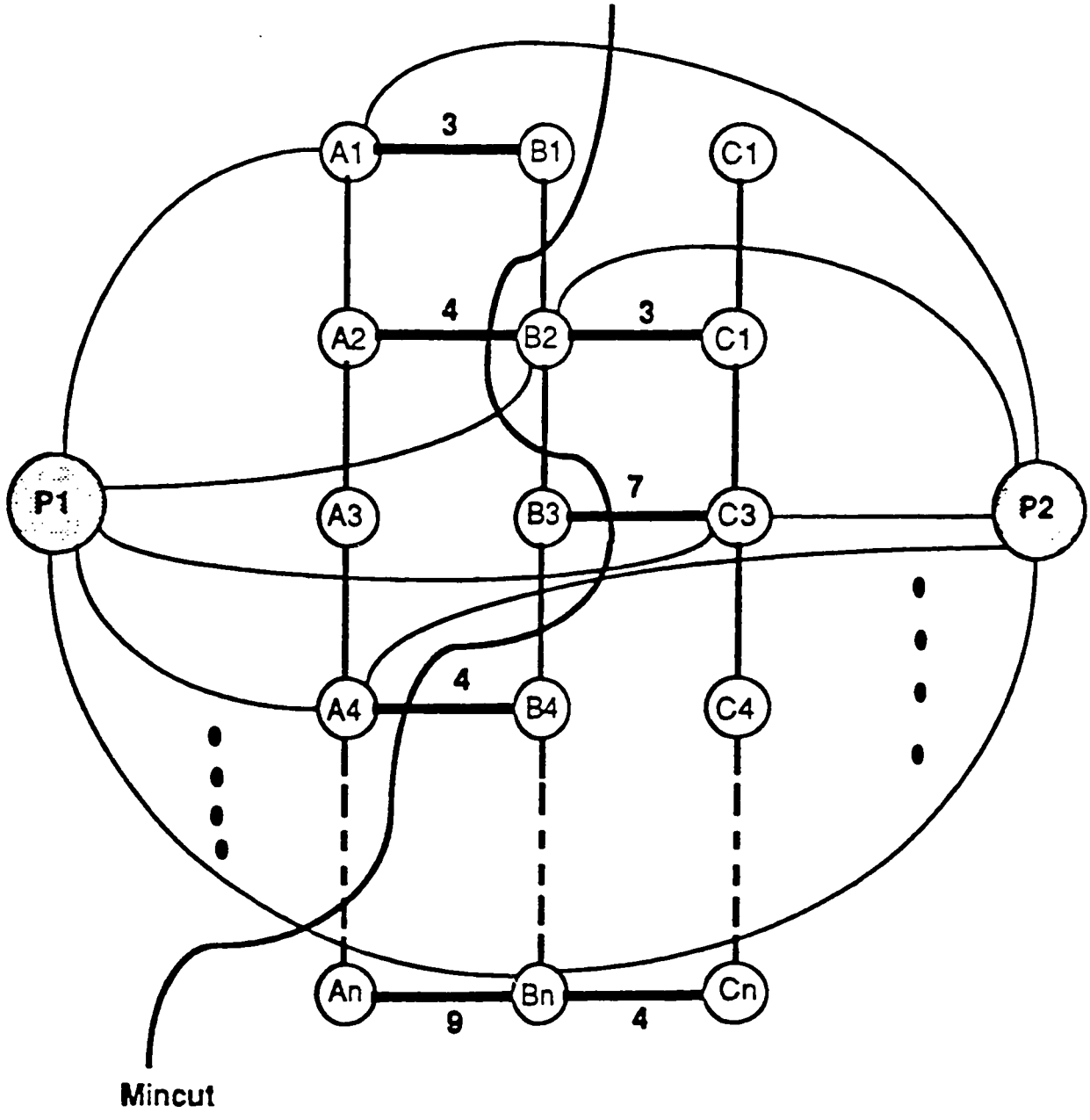


Figure 2.8: A Dynamic Assignment Graph.

approach apply here.

CHAPTER 3

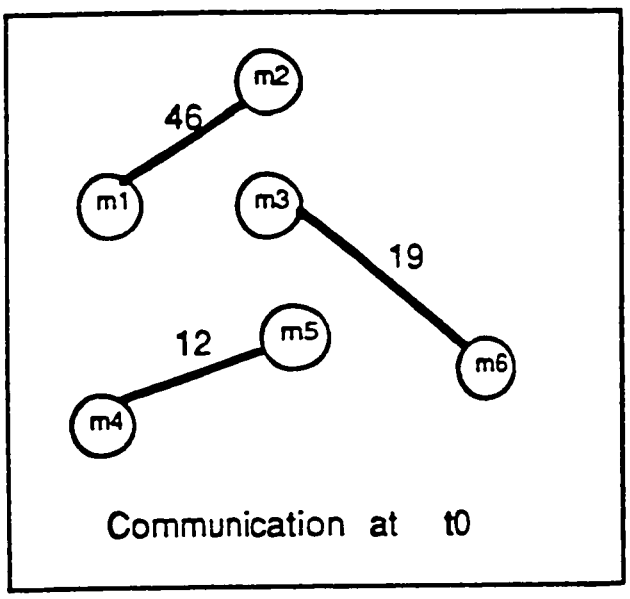
A LOAD BALANCING ALGORITHM

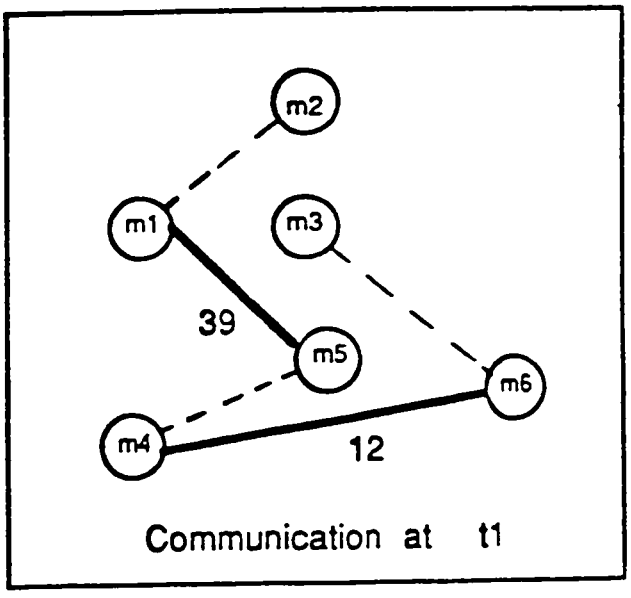
3.1 Problem Statement:

Assigning programs to be executed on a DCS is one of the difficult challenging problems that is being tackled by a large number of researchers in the world, and it is ,indeed, one of the hottest areas these days. However, most of the algorithms published, as presented in chapter 2, exclude concurrency in their models [STON77,STON78,BOKH79,BOKH81]. That is, they assumed that only one module is active at a time and control is passed from one module to another. Whereas, the most important objective of having distributed systems is to execute programs concurrently so that application turn-around time is minimized. Although, the concurrency problem has been considered by some researchers [LEE87, LO88], these algorithms apply for very restricted cases. Lee and Aggarawal [LEE87] assumed that the number of modules (processes) is restricted to be equal to the number of processors and they propose various objective functions that exploit the parallelism as discussed in 2.1.5. Their

algorithm can not be used if the number of processes is greater than the number of processors. Moreover, Lo considered concurrency as a secondary OF in terms of the concept of interference costs [LO88] but concurrency is not considered as a direct goal. She added this cost to allow CPU bound modules that are not communicating with each other to be assigned to different processors so that they can be executed in parallel.

Most applications can be partitioned into a number of modules to be executed concurrently on a set of processors. Partitioning techniques are used to partition the whole application into a number of sub-applications (modules) to be executed in parallel. A good partitioning technique is one that allows more parallelism to take place resulting in potential decrease in application total execution time or resulting in great throughput [AGR88]. Such applications, once started with a set of communication and execution requirements can be subject to transient module communication and processing requirements.To understand what is meant by this statement, let us consider Figure 3.1. This figure shows the communication and processing requirements of an application at two different times t_0 and t_1. The application consists of modules $m_1,\ldots,m_6$.





Execution Costs at t0	
Module	Cost
m1	20
m2	16
m3	5
m4	9
m5	17
m6	12

Execution Costs at t1	
Module	Cost
m1	5
m2	0
m3	0
m4	20
m5	3
m6	18

Figure 3.1: Requirements of a Two Phases Application.

At time t_0, module m_1 is communicating with m_2 and the cost of this communication is 46 units. However, at time t_1,m_1 is not communicating with m_2 it is communicating with another module, m_5 , and the cost of communication is 39. Also processing cost of m_1 is different at different times. Moreover, modules m_2 and m_3 terminated before t_1 (i.e they are not active at t_1). In general, we say that module requirements (wether communication, execution, memory need or any other requirement) can vary from time to time during the lifetime of an application. As a consequence, the balanced assignment that might have been existed at one time may not still hold at another time.

Instead of assuming that processing and inter-module communication requirements be fixed during the life time of an application, as assumed by the classical approaches presented in chapter 2 [STON77,STON78,EFE82,LO88], it is here assumed that these requirements may vary. In our work, we will consider reassigning modules dynamically during program execution so that load of all processors is balanced in time. Module reassignment is done after every contiguous period of time called a 'phase' during which module communication and execution requirements remain fixed.

Assuming that processors are similar, an algorithm that considers changes in execution and communication requirements in time is needed . It decides on which modules should be relocated from phase to phase as the time advances. We believe that the dynamic relocation will impose great reduction in total execution time of the application, and will be better than using static assignment algorithms. Actually, applications that last long times, such as real-time process control, are good candidates for dynamic load balancing.

In section 3.2 we will suggest a heuristic solution to the dynamic load balancing problem with no restriction on the number of processors. In the next section the dynamic load balancing problem is modeled in its general form (i.e for m modules and n processors).

3.2 Mathematical formulation for the dynamic reassignment problem:

Based on the problem statement presented in Section 3.1 a mathematical model for the dynamic reassignment problem is developed. The basis for this model is the phase concept. The phase of an application is defined as a contiguous period of time during which communication and execution requirements remain fixed. It is assumed that modules may be moved between phases only. No module is relocated during a phase. Each phase of the application is associated with the following information:

(1) Execution cost of each module.

(2) Inter-module communication costs.

A formal description of the problem will follow.

Let an application be represented by a number of problem graphs $G_{p1}, G_{p2}, \ldots, G_{pk}$ one graph for each phase. Here k is the number of phases. Each problem graph G_{pq} of phase q is defined as :

$Gpq = \langle V_{pq}, E_{pq} \rangle$,

$V_{pq} = \{v_{pq,1}, \ldots, v_{pq,hq}\}$ a set of problem nodes,

where h_q is the number of modules at phase q.

$E_{pq} = \{e_{pq,i,j}\}$ a set of problem edges for phase q specifying the existence of communication between modules i and j ,where $i,j = 1,2,\ldots\ldots, h_q$.

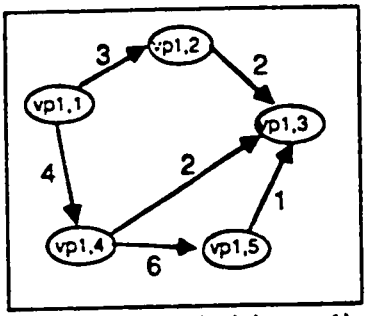
Let $X_{q,i}$ be the cost of executing module i in phase q, where $1 <= i <= h_q$. Also, let $C_{pq,i,j}$ be the weight of problem edge $e_{pq,i,j}$ at phase q. We can interpret the weight of the problem edge $e_{pq,i,j}$ as the number of pakets to be transmitted from $v_{pq,i}$ to $v_{pq,j}$. An important point that has to be mentioned here is that all modules in a phase are assumed to be executed in parallel and hence communication between all modules is activated in parallel.

Let $G_s = <V_s,E_s>$ be a system graph with a set V_s of processing elements $v_{s1},v_{s2},\ldots,v_{sn}$ where n is the total number of processors. E_s is the adjacency matrix that indicates the set of serial communication links in the system used. We restrict the system graph to be fixed during the application life; no nodes or links can fail.

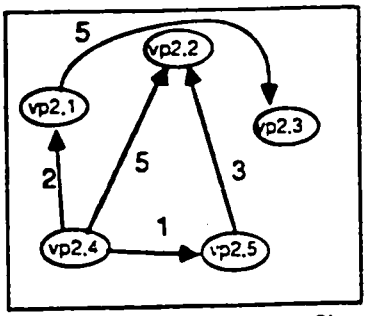
But, in reality the system graph too may dynamically change with processors and/or links added and/or removed. To simplify our problem we have the fixed system graph restriction in our model. Figure 3.2 shows an example of an application with 3 problem graphs (corresponding to 3 phases), and a system graph.

The OF that is employed in our model is to minimize execution and communication in every phase. The unit of costs here is assumed to be time units (e.g msec, seconds, minutes, hours ... etc). Since we assumed that modules are executed in parallel in each phase q, the last communication edge that finishes after all edges determines the cost of communication (CC_q) of that phase.

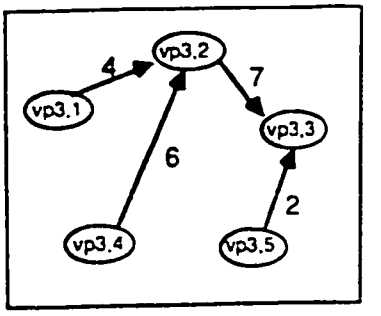
The cost of execution (EX_q) at phase q is the time taken for the processor that finishes after all other processors. The communication cost evaluation procedure is presented as follows. The communication overhead of a problem edge mapped uniquely onto r system links is equal to the time of transmitting the message corresponding to the problem edge through one system link multiplied by r, because the message has to go through r system links; the nominal distance between two system nodes is equal to the number of intermidiate links between the two nodes. However, the



Problem graph (phase 1)



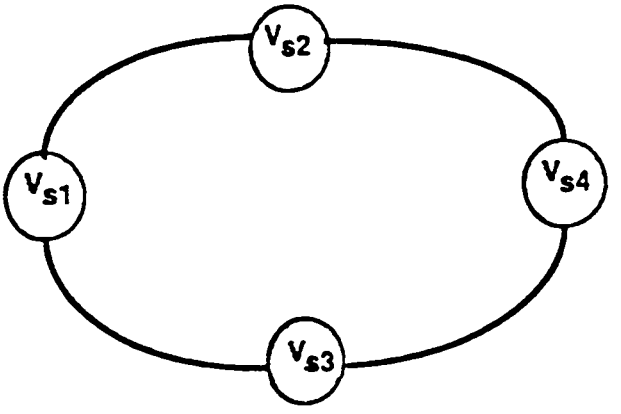
Problem graph (phase 2)



Problem graph (phase 3)

Execution Cost					
Phase 1		Phase 2		Phase 3	
module	Cost	module	Cost	module	Cost
$v_{p1,1}$	10	$v_{p2,1}$	55	$v_{p3,1}$	3
$v_{p1,2}$	9	$v_{p2,2}$	34	$v_{p3,2}$	17
$v_{p1,3}$	35	$v_{p2,3}$	10	$v_{p3,3}$	29
$v_{p1,4}$	26	$v_{p2,4}$	67	$v_{p3,4}$	31
$v_{p1,5}$	11	$v_{p2,5}$	23	$v_{p3,5}$	19

(a) Problem graphs and execution costs



(b) System graph

Figure 3.2: A Sample Application.

nominal distance between the source of a message and its destination can not be used directly for calculating the communication overhead of problem edges being mapped onto system links, because a system edge(equivalently link) can be required by more than one problem edge at the same time. Therefore, the number of communication steps required by a problem edge need to be counted accurately according to the communication protocols of the system [LEE87]. The term communication step is defined as the time needed to transfer one packet through one system link (see Section 2.1.5). A matrix called, CM_q , is used to hold the step count of all problem edges at phase q.

Formally, CC_q and EX_q are defined as :

$$CC_q = \max(CM_{qi,j}), \text{ for all } i,j\text{'s, and}$$

$$EX_q = \max_{p=1}^{n} \{\text{Sum}(X_q\text{'s of modules assigned to processor } p)\}.$$

Knowing the way of calculating execution and communication costs of a given module assignment for phase q, we state that the measure (say OF_q) used for weighting

this assignment is computed as:

$$OF_q = CC_q + EX_q.$$

The objective of our algorithm is to find assignments that minimize OF_q for all phases q, q=1,2,...,k.

3.3 The Load Balancing Algorithm:

Based on the present state of research the solution of the static load balancing problem is found to be a member of the class of NP-complete problems for n > 3. Clearly, the dynamic case injects more complication to the problem because the complexity of this model is higher than the complexity of static case. The number of possible assignments of an application with m modules and k phases to a system that contains n processors is found to be n^{km}, while for the static case it is n^{m}. In other words, the number of modules to handle for the dynamic case is m*k while the number of modules to handle for the classical static case is m. So, we will concentrate on developing heuristic approaches to the load balancing problem with dynamic reassignment. Heuristic approaches have the advantage of achieving reasonable (near optimal) solutions in a short time; it may happen, in some cases, that they obtain optimal solutions. Definitely, a good heuristic solution for the above problem would be an important contribution in this field.

In this section we present a heuristic algorithm to the problem presented in 3.2. The application is represented by problem graph (one graph per phase) and execution cost

vectors (one vector per phase) that define the communication and execution requirements, respectively, are needed by the application in each phase. Our algorithm will consider every phase separately and map the application modules of that phase onto processors such that this mapping is suboptimal (i.e. the OF for the mapping is minimized approximately). The reassignment is carried out after the mapping algorithm for the next phase is complete. Note that the algorithmic problems faced in a phase are those faced in a nonphase static mapping problem.

Let a phase have n processors and m modules. The proposed solution is handled in two main steps. First, an efficient graph reduction(or module clustering) technique is used to reduce the size of the problem graph to system size if m>n. This is done by coupling some modules according to a certain criterion until m=n. On the other hand, if m<n, dummy modules are added so that n=m. For the rest of discussion we assume that m >= n because the case where m<n is straightforward to handle.

The general graph reduction problem is an NP-complete problem. Hence, a heuristic solution is proposed. This solution is based on a factor called the "coupling degree (cd)" applied to every pair of modules in a phase ,say, q.

The modules in a pair are linked by a problem edge.The coupling degree of a pair $(V_{pq,i}, V_{pq,j})$ is defined as

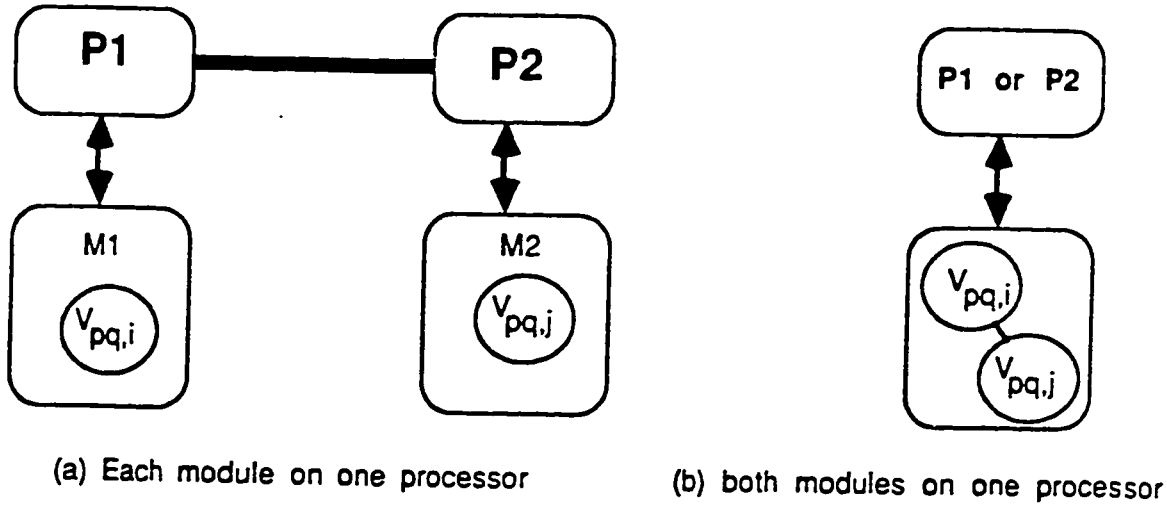
$$cd_{ij} = \{\max(X_{q,i}, X_{q,j}) + C_{q,ij}\} - \{X_{q,i} + X_{q,j}\}$$

This concept is based on the possible ways of assigning the two module, $v_{pq,i}$ and $v_{pq,j}$, to two processors v_{s1} and v_{s2} as shown in figure 3.3 .

There are two cases for assigning the modules. Either each module is assigned to a separate processor (Figure 3.3(a)) or both are assigned to the same processor (Figure 3.3(b)). The cost of the first case is equal to $\{\max(X_{q,i}, X_{q,j}) + C_{qi,j}\}$, because $v_{pq,i}$ and $v_{pq,j}$ are executed in parallel. However, the cost of the second assignment is equal to $\{X_{q,i} + X_{q,j}\}$ because they are executed by one processor, and because intra-processor communication is assumed to be zero. The difference between the two costs gives the value of coupling degree, cd_{ij}.

The coupling of the two modules is decided by the following procedure:

1. Let M be a set of m clusters where each cluster contains one module.
2. Let S be a set that contain possible pairs of modules



(a) Each module on one processor

(b) both modules on one processor

Figure 3.3 : The possible ways of assigning two modules

$(v_{pq,i}, v_{pq,j})$. to be considered. Initially, S is empty. For each edge $e_{pq,ij}$ in the problem graph we add a pair $(v_{pq,i}, v_{pq,j})$ to S until all edges are exhausted.

3. Evaluate cd_{ij} for all pairs in S.
4. Order S according to their cd_{ij}'s in descending order .
5. Couple the modules contained in the pair that has the maximum cd_{ij} and regard them as a single new module (new cluster).
6. Remove this pair from S.
7. Refine the edges connected to the new cluster by linking the edges connected to the individual modules to the cluster they are in.
8. Set the execution cost of the new cluster to be equal to $(X_{q,i} + X_{q,j})$.
9. Update the cd of the edges connected to the cluster and decrement m by one.

Steps 4-9 are repeated until m = n.

Upon completion of the graph reduction, we will have n clusters, and each cluster may contain one or more modules. Each cluster is now considered as a single new module with new execution and communication weights of the constituting modules. Most of the time taken by the clustering method is

due to the sorting of edges. Using an efficient sorting algorithm like heap sort the complexity of sorting is O(e log e) where e is the number of edges in the problem graph. Since steps four to nine are executed (m-n) times, the overall complexity of the whole procedure is O((m-n)e log e). If m >> n, the complexity is approximately O(me log e).

The problem now is reduced to a mapping problem with n composite problem nodes and n system nodes. The second step is to find an initial assignment based on a certain heuristic criteria; like graph homomorphism considering a number of factors ,namely communication intensity of problem nodes, the degrees of both problem nodes and system nodes, and the neighborhood of nodes[LEE87]. If the resulting OF of the mapping achieved by the initial assignment procedure is not satisfactory, we refine the assignment using the pairwise exchange procedure. However, we include the execution cost in evaluating the OF (see Section 3.2). Upon termination of the second part of the load balancing algorithm, the suboptimal mapping of a phase is complete. The above proposed solution is applied to each phase.

So far, assuming that $m \geq n$, our solution to the problem reduces the problem graph into the size of the system graph after which the reduced graph is mapped into the system

graph using Lee's approach. Obviously, the mapping produced will result in utilizing all processors of the DCS to execute the modules of a phase. But, there can be problem graphs that can be mapped into a subset of the processors and OF_q's of which are less than that of mapping those problem graphs into the whole set. This situation can be covered by the following modification to the algorithm. The problem graph is reduced into i clusters, i=1,2,3,...,n. For all values of i, the reduced graph is mapped into the system graph using the same mapping algorithm. We select the mapping that has the smallest OF_q among all i's. In Chapter 4, the algorithm implementation is presented with sufficient details.

The time complexity of the load balancing algorithm is determined by, the graph reduction, the initial assignment, and the pair-wise exchange procedures. The complexity of these procedures are O(me log e), $O(n^2)$ [LEE87], and $O(N_i D_M \alpha_M)$ [LEE87], respectively, where e is the number of edges, m is the number of modules, n is the number of processors, D_M is the diameter of the system graph(i.e the longest path), α_M is the maximum edge weight in terms of the number of packets in the problem graph and N_i is the number

of iteration in the pair-wise exchange. Since we have to apply all three procedures n times, the overall complexity is $O(mne \log e + n^3 + nN_i D_M n_M)$.

3.4 Theoretical Enhancements to the Load Balancing Algorithm

So far we did not give attention to the cost of relocating modules from phase to phase. Clearly, the gains from relocation must not be smaller than the cost of bringing the relocation cost into picture. The next two sections present two solutions to the problem. Before presenting the solutions, we show how to evaluate relocation cost between two phases.

Knowing the assignment of each module in two successive phases q and q+1, we can define the set of modules that can be the candidates for relocation just after phase q. Let us call this set RS_q. In addition to the assumptions of Section 3.1, we assume that relocation of modules between phases take place simultaneously. So, the cost of relocation (R_q) can be evaluated similar to evaluating the communication cost. That is,

$$R_q = \max(RM_{q,i})$$

where $i=1,2,\ldots,|RS_q|$, and relocation matrix , RM, entries are evaluated in the same way as evaluating the entries for communication matrix, CM. Thus, the cost of running and relocating the application in phase q becomes:

$$OF_q = CC_q + EX_q + R_q.$$

3.4.1 Integral Suboptimal Solution over all Phases:

The main implication of this approach is that the individual phase mappings are not necessarily optimal. However, the aim is to get the mapping over all the phases be optimal. As stated before, calculation of relocation costs need an advance knowledge of the suboptimal mappings at each phase, separately considering only communication and execution costs of modules. We will use the method of section 3.3 for achieving suboptimal mappings for each phase independent of other phases without considering relocation costs. With the knowledge of these mappings we are able to identify the set of modules to be relocated after each phase and evaluate their relocation costs as shown above. The integral suboptimal approach that we suggest to minimize the total cost of the entire application is restricted to the following : either we relocate all candidate modules or relocate none (i.e no partial relocation from phase to phase). The total cost of an application (COST(A)) with full relocation is defined as

$$\text{Cost}(A) = \sum_{q=1}^{k} (CC_q + Ex_q + R_q)$$

This can be done efficiently by constructing a directed graph called "global relocation graph" with nodes

corresponding to mappings and levels corresponding to phases(Figure 3.4). The source node (S) of this graph is the mapping of the first phase where no other possible mappings exist; because there is no relocation for the first phase. The nodes that reside in the same level correspond to the possible mappings in that phase. Each node in phase i branches to two new nodes indicating two alternatives in the next phase. The left node is associated the suboptimal mapping of phase i+1,i.e. full relocation; and the right node corresponds to a no change state, i.e. no relocation. Furthermore, the left edge has a weight equal to execution and communication cost of the mapping at phase i (EX_i+CC_i) plus relocation of modules from phase i to phase i+1 (R_i). The right edge has a weight equal to execution and communication cost (EX_i+CC_i) of the mapping at phase i only; because relocation cost is zero. To find the best mapping among all phases which minimizes the total cost, we employ a shortest path algorithm between the source node of the graph and the node that resides in the bottom-most level. The latter node is a dummy node and the edges connected to this node have zero weights. Note that the weight of a path in the graph is equal to the cost of the mappings along the path, and hence the shortest path will correspond to the

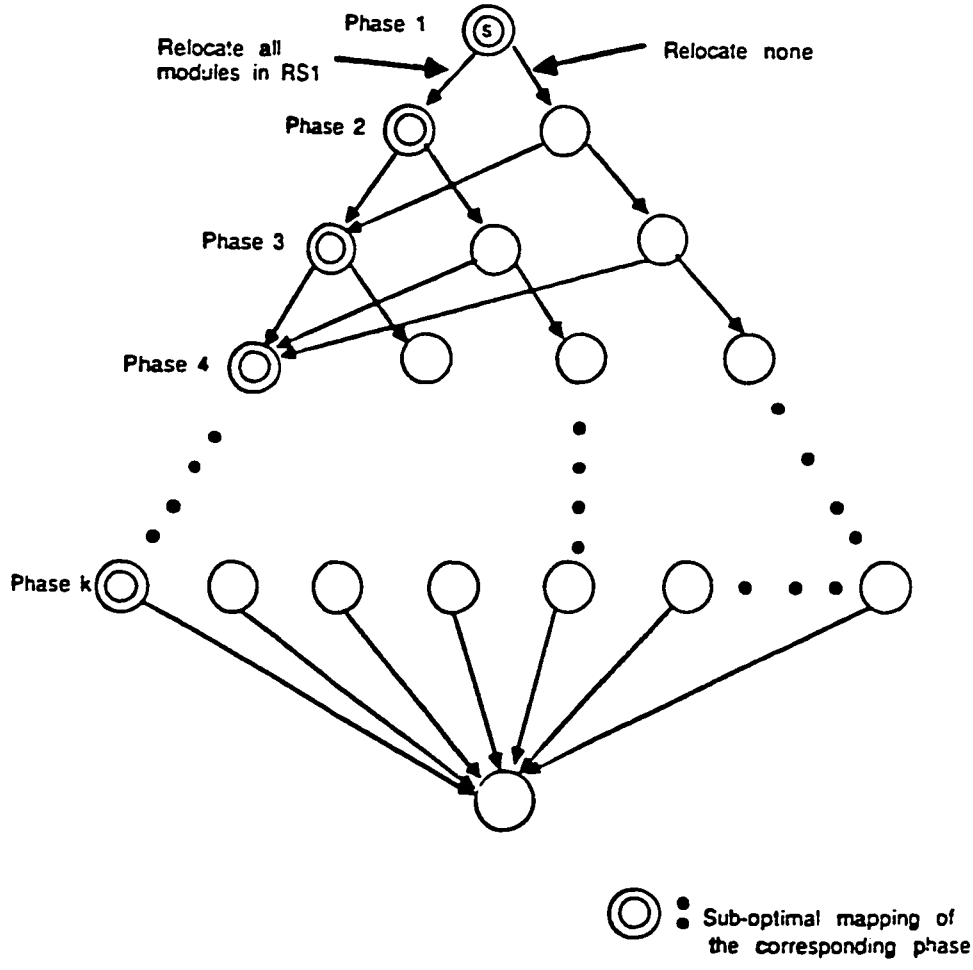


Figure 3.4 : Global Relocation Graph

minimum cost. The number of edges in this graph is K^2 and the number of nodes, excluding the last node, is

$$\frac{K^2+K}{2}.$$

Since we need to evaluate the OF for each edge we have to multiply the complexity of evaluating OF by K^2 and add to it the complexity of the shortest path algorithm. The complexity of this algorithm is determined by the complexity of evaluating communication cost ($O(nD_M \alpha_M)$), relocation cost ($O(nD_M \alpha_r)$), and execution cost($O(m)$). The term α_r is the size of the largest application module in terms of number of packets . The other terms are already defined in section 3.3. Thus, the total complexity of the algorithm is $O(K^2(nD_M \alpha_M + nD_M \alpha_r))$.

So far we considered restricted relocation (i.e full or no relocation). There are a total of n^m possible relocations between two phases. Therefore, obtaining optimal solutions globally even for two phases is NP-hard for which an efficient solution may not even exist.

In the next section, we describe an integrated local suboptimal solution to the partial relocation problem. Local approach assumes that the individual phase mappings

are optimal.

3.4.2 An Integrated Local Solution:

In real time environment the integral optimal approach is not practical because of its time and space complexity. Assuming that the duration of the phases are long enough, the mapping of each phase can be handled independent of the global mapping. The mapping in a new phase is considered to depend on the previous phase only. In real time environment this approach may be considered as realistic. This consideration is supported by the following reasons:

- Global communication requirements as well as processing requirements cannot be predicted well before the application has even been executed, although many methods are proposed for requirement estimation at compilation time [CHU80].
- In many dynamic real time applications, the transition between adjacent phases, the data related to communication and processing requirements of the two phases can be known to a great precision.
- The locality concept of distributed programs states that the set of active modules remain active for some time and changes slowly during the duration of the application. The behavior of the module set,

here, is similar to the working set of a serial program.

The problem is thus reduced to adjacent phases. The suboptimal mapping of the current phase is normally known. The mapping for the next phase is computed from the new application data and the system data. The solution proposed to this problem is based on constructing a graph similar to the one given in Figure 3.5. The source node (S) of this graph is the mapping of the current phase with n links. Each link indicates relocation of m_1 to one of the n processors. The second level of the graph corresponds to all possible relocations of m_2, the third level corresponds to all possible relocations of m_3,.... and so on. We call this graph "Reassignment graph". The approach used to find the best mapping is to apply a shortest path that corresponds to the optimal mapping for the transition between two adjacent phases. Every path from s to node d corresponds to a unique assignment. Evaluating the OF of a path is similar to the cost function ,Cost(A) evaluation presented in section 3.4. The number of node of a reassignment graph for an application with m modules and n processors is O(mn). The total number of paths in the graph is exponential and, hence, the complexity of this approach is also exponential;

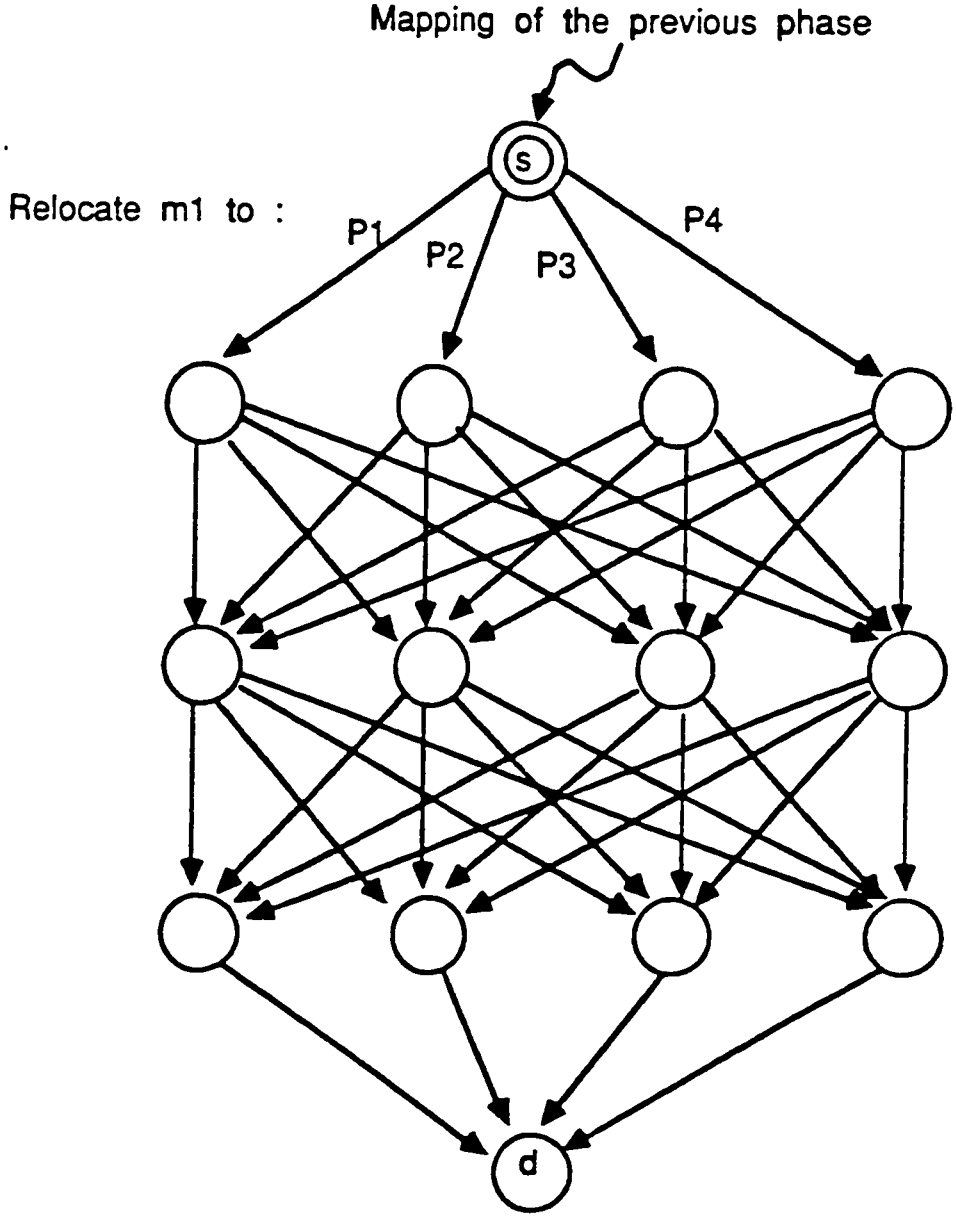


Figure 3.5 : Reassignment Graph for 3 modules and 4 processors.

i.e. NP-hard. Suboptimal solutions are required.

3.5 Other Suboptimal Solutions

Because the above two algorithms are impractical, we suggest another strategy for relocating modules. This strategy assumes that the number of phases is large. We use the algorithm described in 3.3 to obtain suboptimal mappings for the individual phases. Based on comparing an estimate of the total relocation cost to both execution and communication costs of an application, we follow one of tw policies. If relocation cost is insignificant (i.e. very small compared to execution and communication costs), we force modules to be relocated in every phase according to the mapping of that phase. On the other hand, if relocation cost is significant (either comparable or greater than execution and communication costs), we follow another strategy. Before initializing the first phase, we broadcast copies of all modules to all processors so that every processor has a copy of all modules. After that, in every phase, a central module issues control messages to all processors to activate the modules assigned according to the mapping of that phase. This policy will reduce relocation cost substantially, because we have assumed that the number of phases is large, and it saves the time of executing

an algorithm that decides which module should be relocated and where. This approach, however, ignores module creation or termination during intermediate phases.

CHAPTER 4

IMPLEMENTATION ASPECTS OF THE LOAD BALANCING ALGORITHM

4.1 Introduction

The objective of this Chapter is to give a detailed implementation of the load balancing algorithm described in Chapter 3. A Pascal program was written to implement the load balancing algorithm. This program is invoked for every phase of an application. It takes as input the application requirements (execution and communication) of that phase to determine the suboptimal assignment of modules. The flow diagram shown in Figure 4.1 describes how the program works. The program starts with Initialize procedure which will read the application requirements (problem graph and execution table) and system topology. After executing the Initialize procedure, Clustering procedure is called n times to reduce the problem graph into sizes 1,2,3, ... and n, in order to produce n reduced problem graphs $(PS_1, PS_2, \ldots, PS_n)$, respectively. Then, each one of these reduced problem graphs is mapped into the system graphs by cluster-map procedure

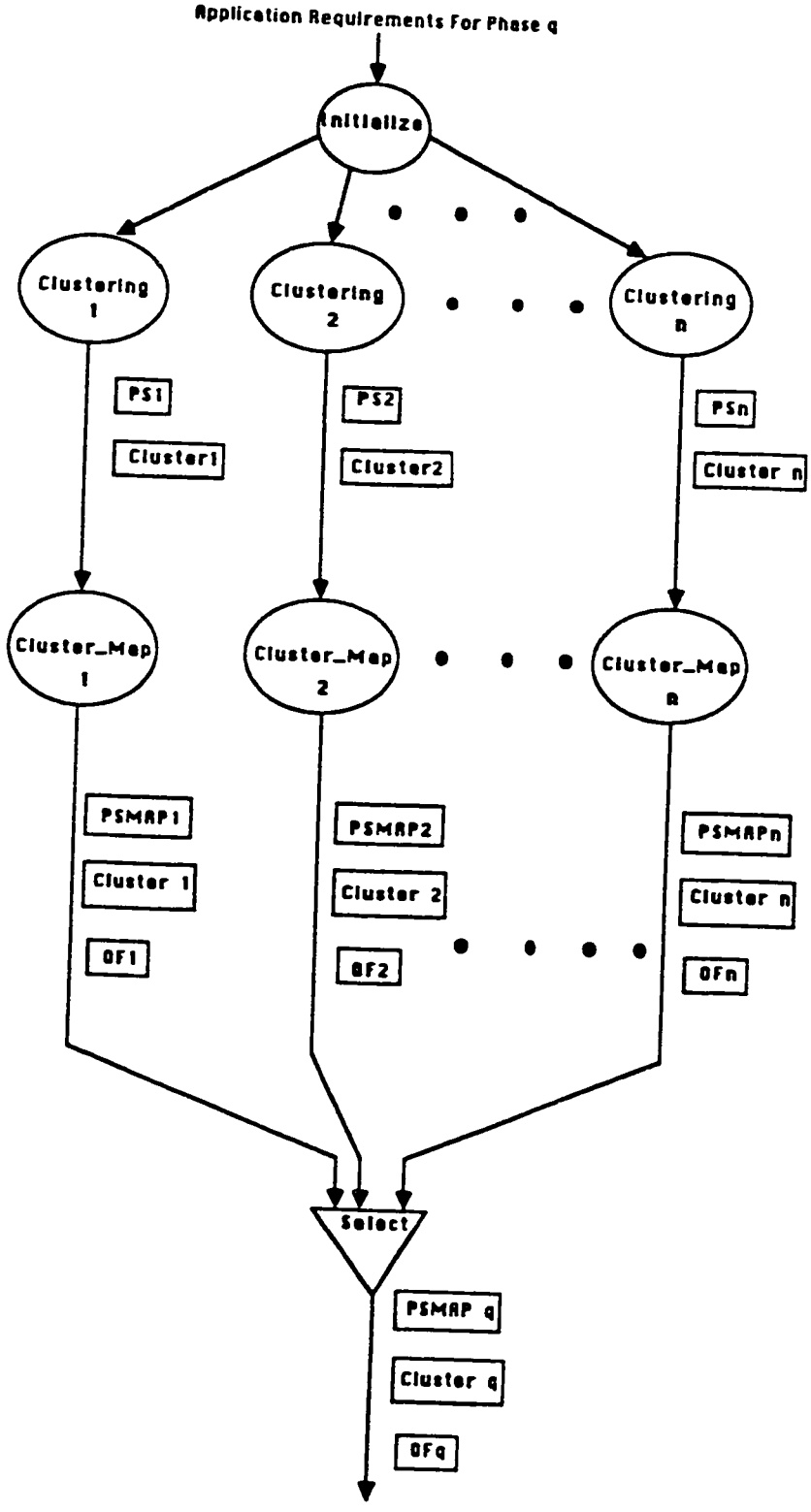


Figure 4.1 : Flow Diagram of the Program.

producing a suboptimal mapping for the corresponding problem graph. Cluster-map procedure achieves the suboptimal mapping in two steps. In the first step an initial mapping (or assignment) is obtained using the graph homomorphic technique presented in Chapter 2. As a second step, a better mapping is obtained from the initial mapping using pairwise exchange approach. Also, the cluster-map procedure computes the objective function for all the n mappings of the n reduced problem graphs (i.e. OF_1 is the objective functions of the mapping for PS_1,OF_2 is the objective function for PS_2, and so on). The mapping that has the smallest OF among all of the n mappings is selected as the suboptimal mapping of phase q. In Section 4.2 the implementation details of procedures Initialize, Clustering and cluster-map are presented.

4.2 Procedure Details

4.2.1 Initialize Procedure

Initialize procedure is given in Figure 4.2. The function of this procedure is to read the specifications of the application and the system. The application specification includes number of modules (m), edge weights

Procedure : Initialize

```
Purpose : (1) Reads Problem and System graphs,and
              execution table.
          (2) Assigns heaps to every system link.
          (3) Computes Routing matrix (R),and
              Degree matrix(D).
Input : m = number of modules
        n = number of processors
        S = number of modules(nxn)
        P = Problem Edge weight Matrix, it represents the
            problem graph (mxm)
       XC = Execution Cost Matrix (m)
Output: D = number of modules
        R = Routing Matrix
       EM = Edge to Heap mapping matrix (nxn)
```

```
BEGIN
        Read m,n,S,P,XC
        Compute D and R using a shortest path algorithm
        For i = 1 to n Do
           For j = 1 to n Do
           BEGIN
              IF S[i,j] = 1 THEN
              BEGIN
                Assign a unique heap for link S[i,j]
                EM[i,j] = ¬Heap
              END
           END
END
```

Figure 4.2 : Initialize Procedure

of the problem graph in the form of (mxm) matrix (called P), and execution cost of all modules in the form of one dimensional array (XC). The system is specified by the number of processors (n), and the processor connectivity matrix S. In addition to reading these matrices, Initialize procedure constructs three matrices to be used by other procedures namely D,R,EM. The D matrix is an nxn matrix which contains the nominal distances between any pair of system nodes (i.e. the number of intermediate links between every two processors). The R matrix is a routing matrix (nxn) used in routing messages. Computing the R matrix is done using a shortest path algorithm. However, we don't have to restrict ourselves to the shortest path. Another routing strategy may be used. This procedure also assigns heaps to all system links and makes EM point to these heaps. Heaps are used to store packets, that need to be transmitted through a link, in a particular order. The advantage of using heaps is given later.

4.2.2 Clustering Procedure

Clustering procedure is responsible of coupling the proper modules to reduce the problem graph into the size of the system graph or any other size less than n. This procedure is given in Figure 4.3. It is an exact

Procedure : Clustering

```
Purpose: Reduces a problem graph of size m into a graph
         of size n.
Input : m = number of modules
        n = number of processors
        P = Problem Edge Weight Matrix (mxm)
       XC = Execution Cost Matrix (m)
Output:PS = the reduced problem matrix of P.
          Cluster = module to cluster mapping (m)
Local: E-list = doubly linked list to store module pairs
       Edge-loc  = Pair location in E-list (mxm).
       cd        = coupling degree
       csize     = number of clusters
```

```
BEGIN
  IF n= 1 then
    Cluster[i]=1 for all i=1,2,..,n (*only one cluster*)
    PS[i,j] = 0 for all i,j
  ELSE
    Cluster[i] = i for all i ≤ m
    csize = m
    PS    = P
    For i = 1 to n Do
       For J = 1 to n Do
       BEGIN
       cd=PS[i,j]+max(XC[i],XC[j])-(DC[i]+XC[j])
       insert(pair(i,j), cd) in E_list and make
       Edge[i,j] point to its location.
       END
(* actual clustering *)
For i = 1 to (m-n) do
Begin
  (*Remove the first pair (k,l) from the E_list;
    pair (k,l) has the maximum cd *)
      Remove (k,l).
      Combine modules (k and l) in one cluster(i.e.
      update Cluster).
      csize = csize - 1
      Remove all problem edges connected to either K or L
      Recompute cd for these edges (* XC of the new
      cluster = XC[k]+XC[l] *)
      Reinsert these edges with the new cd in E_list.
      Update PS
end;
End.
```

Figure 4.3 : Clustering Procedure

implementation of the graph reduction algorithm in Section 3.3. A reduced problem graph (called PS) of the problem graph P is generated. Also, module clusters is returned by this procedure in the form of a one dimensional array (Cluster) used to represent the module to cluster mapping(e.g. module i is in cluster Cluster[i]). If our aim was to map all modules into a single processor (i.e. n=1) then communication cost would be zero and all entries of PS would have been set to zero. On the other hand, if n ≠ 1 then we need to compute the coupling degree (cd) of each communicating module pair (i,j). The coupling degree of the a pair is computed as

cd[i,j] = P[i,j]+max (XC[i],XC[j])-(XC[i]+XC[j]).

Two other procedures are called by this procedure : Insert and Remove. Insert is called after each computation of the coupling degree of a pair to insert it in a doubly linked list (E-list) such that the content of the E-list is sorted in descending order according to the cd of pairs after each insertion. An array called Edge-loc is used to hold the address of a pair in the E-list. The advantage of Edge-loc is to make removing any pair from the E-list very simple; in other words no searching is needed.

After ordered insertion of all pairs, an iterative reduction process is started. In every iteration modules of

the pair that reside at the front of the E-list are combined into one cluster (or a new module) after which the pair is removed from the E-list by Remove procedure. Also, all edges connecting any module node of the pair to another module is removed and inserted again with new coupling degree. The reason for this re-insertion is due to the change of the execution time of the new module resulted from joining the two modules in one cluster. PS, which is initialized as a copy of P, is updated to reflect the clustering. The reduction process stops when the size of the reduced graph equals the size of the system graph or the required size(i.e. < n) The resulting clusters and the reduced problem graph (PS) are returned by this procedure.

4.2.3 Cluster-Map Procedure

This procedure is basically an implementation of the pair-wise approach presented in Chapter 2 using an objective function for parallel processing (OF_2) [LEE87]. One of the important extensions to this procedure is that we assume that all links can be used in parallel to send packets. Yet, another extension is the use of heaps to store packets of a system node to be sent through a system link. The advantage of heaps is to make the OF calculation more

accurate by efficient sorting of packets in the heap according to the product of the number of packets of the problem edge to whom the packet belongs and the nominal distance between the current system node and the destination of the packet. By this way we give longer messages that have to go through greater number of communication links the priority for sending. Figure 4.4 shows the flow diagram of Cluster-map. As we see in the figure, this procedure contains three procedures that cooperate together to obtain a suboptimal mapping.

4.2.3.1 Initial Assignment Procedure

The objective of Initial-Assignment procedure is to try to obtain an initial suboptimal mapping. The basis of this procedure is graph homomorphism of the reduced application graph and system graph. The procedure shown in Figure 4.5 is presented in Pascal like code. The input to this procedure is a reduced problem graph (PS) and module clusters; and the output is the initial mapping of the clusters in the form of a one dimensional array, PSMAP (subscripts represent cluster number and entry content represents the system node to which a cluster is mapped).

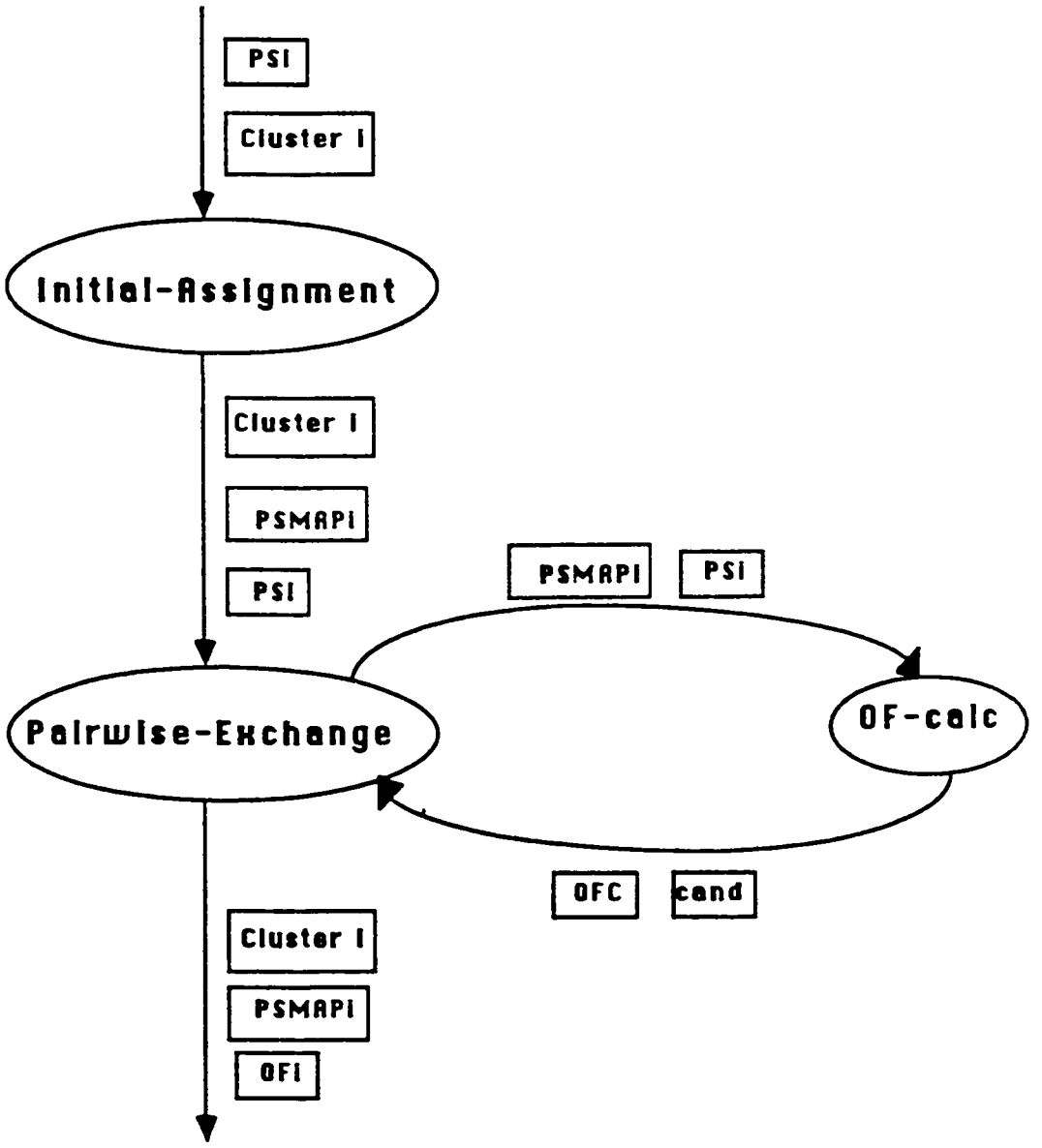


Figure 4.4 : Flow Diagram of Cluster-Map Procedure.

Procedure : Initial-Assignment

```
Purpose: Achieves an initial cluster-to-processor
         assignment.
Input  : PS = Reduced problem graph
         Cluster = module to cluster map (to be passed to
                   the next procedure)
Output: PSMAP = Cluster to processor mapping (n)
Local  : CI    = Communication Intensity of clusters (n)
         DEGP  = Degree of cluster nodes (n)
         DEGS  = Degree of system nodes (n)
```

```
BEGIN
        CI[i] = sum (abs(PS[i,j])) for all j ≠ i.
        DEGP[i] = number of problem edges connected
        to cluster i, for all i = 1, ..., n
        DEGS[i] = number of system edges connected
        to node i, for all i = 1, ...,n
        Find a cluster k which has the largest
        communication intensity.
        Assign cluster k to a system node j such that
        DEGP[k] is as close as possible to DEGS[j]:
        PSMAP[k] = j.
        Remove cluster k from the set of cluster.
        While All clusters are not assigned Do
        Begin
          - Find a cluster t with the largest CI from
            the clusters which are already assigned.
          - Assign t to a system node r such that the
            maximum of the nominal distances between
            r and the system nodes that are already
            assigned clusters adjacent to cluster t
            is minimized : PSMAP[t] = r.
          - remove cluster t from the set of clusters
        End;
End.
```

Figure 4.5 : Initial-Assignment Procedure

Initial Assignment starts by initializing three one dimensional arrays: CI (with size m), DEGP (with size m) and DEGS (with size n). CI contains the communication intensity of the clusters being mapped. CI of cluster i is computed as:

$$C\{i\} = \sum_{j=1}^{k} |PS\{i,j\}|$$

where k is the total number of clusters. DEGP contains the degree of a cluster node defined as the number of problem edges coming in or out of the mode. DEGS is computed in a similar way. The rest of code performs an efficient initial assignment using the three arrays. By an efficient initial assignment the number of pairwise exchanges can be minimized. In an ideal case, i.e. optimum initial assignment, no pairwis exchange is needed. Note that this procedure minimizes the communication cost.

4.2.3.2 Pairwise-Exchange Procedure

The Pairwise exchange procedure is shown in Figure 4.6. The basic function of this procedures is to optimize the mapping obtained by the Initial-Assignment procedure. The inputs to this procedure are PS and the initial mapping represented by PSMAP which is obtained by the Initial-

Procedure : Pairwise exchange

```
Purpose: Improves mapping achieved by Initial-Assignment
         using pair-wise exchange.
Input : PS = Reduced problem graph  (nxn).
        PSMAP = Initial cluster to processor assignment(n)
        Cluster= module-to-cluster Map array (m)
Output: PSMAP = Final mapping (n)
        OF    = total cost of the assignment
```

```
Begin
  Repeat
        Call OF-calc (PS,PSMAP,OFC,cand);
        For i = 1 to n Do
        Begin
          Exchange cluster i with cand temporarily.
          Call OF_calc (PS,PSMAP,OFC, cand_i)
        end;
        Determine cluster j that corresponds to the
        smallest OFC
        PSMAP[j] <--> PSMAP[ cand_j]
        cand = cand_j
  Until (no improvement in OFC);
  Calculate OF of the resulting mapping:
        OF = OFC + OFX (* OFX is the execution
        cost of mapping *).
End.
```

Figure 4.6 : Pairwise-Exchange Procedure

Assignment procedure. Initially, this procedure calls OF-calc presented in Figure 4.7 to compute OFC of the current mapping, PSMAP and determines the candidate cluster for exchange. The candidate cluster, cand is one of the clusters to which the problem edge (in PS) with the largest communication overhead connects. This cluster, cand is exchanged with all other clusters temporarily and the OFC after each exchange is evaluated by calling OF-calc. Formally,

PSMAP[cand] <---> PSMAP [j] for all j ≠ cand.

Among these exchanges the exchange with minimum OFC is selected. The new OFC is compared with the old OFC. IF it is smaller the exchange is made permanent and the new OFC replaces the old OFC. The exchange is repeated until the new OFC is larger than the old one. The total cost of the resulting mapping PSMAP is evaluated, and PSMAP is returned as the suboptimal mapping for the phase.

4.2.3.3 OF-Calc Procedure

Given a module to processor mapping (PSMAP), OF-calc calculates the time needed by modules to complete communication (See Figure 4.7). It uses an auxiliary data structure called packet for this purpose. The weight of a

Procedure : OF-calc

```
Purpose: Computes communication overhead of a given
         cluster-to-processor mapping.
Input : PS = Reduced problem graph  (nxn).
        PSMAP  = Cluster to processor mapping (n).
Output: OFC    = Communication cost of PSMAP.
        cand   = candidate for exchange.
Local : Heap = number of one dimensional arrays
        simulating binary trees (to be used in
        heapsort)
```

```
Begin
  C[i,j] = 0 for all i,j ≤ n.
  For all PS [i,j] ≠ 0 Do
     Begin
        Source = PSMAP[i]
        Dest   = PSMAP[j]
        pkt-priority=PS[i,j] * D[Source,Dest]
        Assign PS[i,j] packets to Heap number EM [Source,
                          R[Source,Dest]]
          which corresponds to the link connected
          to system node Source through which the
          packets will be transferred, and store
          the information (source, dest, pkt-priority)
          together with packet
        Sort the packets in the heap in ascending
          order according to pkt-priority using heap
          sort.  So the packet with smallest priority
          appears at the root of the heap.
  end;
  step = 0
  Repeat (*move packets to their destinations *)
    Step = step + 1
    For current = 1 to n Do
      For every heap in node current Do
        Begin
           If heap is not empty then
             Begin
```

Figure 4.7 : OF-calc Procedure

```
            remove the packet that resides at the
            root of the heap and readjust the heap.
            insert this packet in heap EM [current node,
            R[current,Dest]] and readjust the heap.
            (*This step simulates forwarding packet to
              the next node *)
            nextnode = R[current,Dest].
            IF nextnode = Dest then
               C[Source,Dest] = Step
         End
      Until (all packets reach their destinations);
      OFC = the maximum entry in C
      Cand = Cluster i or j where c[i,j] = OFC.
End
```

Figure 4.7 : Of-Calc Procedure (Contd)

problem edge $e_{pi,j}$ can be represented as the number of packets to be sent from m_i to m_j. As stated before the weights of problem edges is stored in P. Each packet is associated information about its source, destination and priority. The packet priority is defined as:

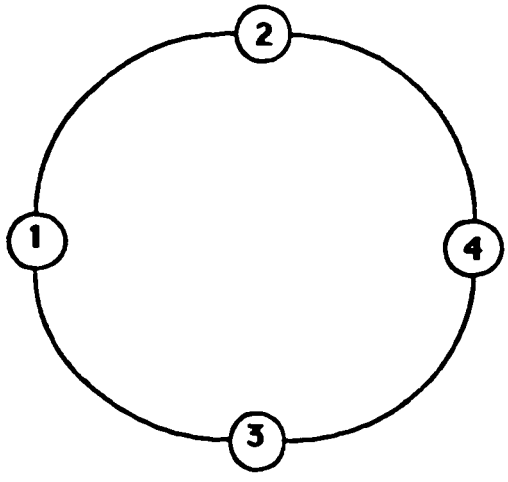
PS[i,j] * D[k,j]

where node i is the source, node j is the destination, and node k is the current node that holds the packet. The packet priority is used to make the objective function more accurate. Every system link is assigned a heap to store the packets that wait for transmission through that link. OF-calc starts by assigning the number of packets on a problem edge i,j, i.e. PS[i,j], to the heap that corresponds to the system node that the cluster (i) is assigned to. The communication cost matrix (C) is used to save the communication cost of every problem edge of the reduced problem graph. Initially all entries of C are set to zero. A packet is removed from every heap and assigned to the next heap. The next heap is determined by the Routing matrix (R) and the EM matrix which stores the heap number for every system link. This process is repeated until all packets reach their destinations. A variable, Step is incremented in every iteration to keep track of the total number of iterations which is returned as the value of OFC. Step is

exactly the number of communication steps needed for all packets to reach their destinations. Also, in every iteration, we check if the last packet from its source reaches its destination. If this happens, the C entry of the corresponding problem edge is set to the current value of step. The last thing done by OF-code is to determine the candidate for exchange. The candidate cluster for exchange is determined as one of the clusters to which the problem edge with the largest communication cost connects.

4.3 A Typical Load Balancing Problem

In order to illustrate how suboptimal solutions are obtained by the algorithm, a typical application with 3 phases is considered. For this application the execution costs of modules and weights of the problem edges are generated randomly. The application consists of 7 modules to be mapped onto the system graph shown in Figure 4.8(a). The processor connectivity matrix, S of this graph is shown in Figure 4.8(b). The problem graphs of the three phase are shown in graphs (Figure 4.9) and Matrices (Figure 4.10). In addition, execution costs XC, of the seven modules, at the three phases are given in Figure 4.11. The steps for obtaining the suboptimal mapping of modules for the first

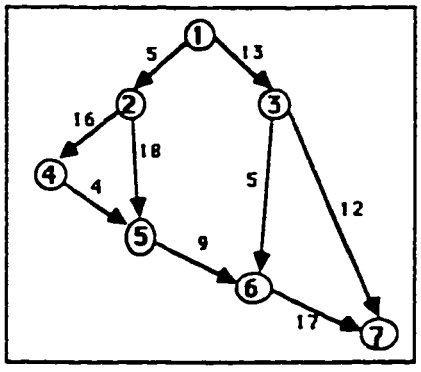


(a)

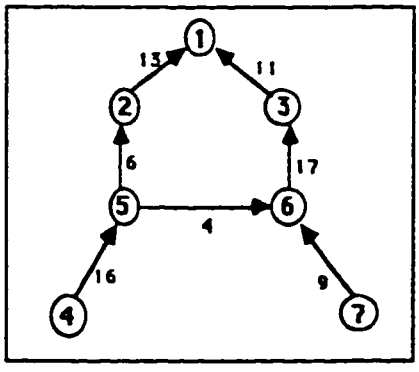
$$S = \begin{vmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{vmatrix}$$

(b)

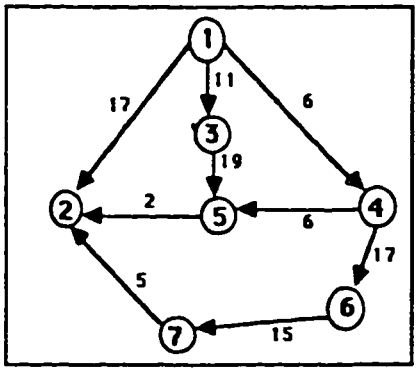
Figure 4.8 : The System Graph Used in the Illustrative Example.



Phase 1



Phase 2



Phase 3

Figure 4.9 : Application Problem Graphs (m = 7).

$$P = \begin{pmatrix} 0 & 5 & 13 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 16 & 18 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 5 & 12 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 17 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Phase 1

$$P = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & 0 & 0 & 0 & 0 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 16 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 4 & 0 \\ 0 & 0 & 17 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 \end{pmatrix}$$

Phase 2

$$P = \begin{pmatrix} 0 & 17 & 11 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 19 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 17 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 15 \\ 0 & 5 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Phase 3

Figure 4.10 : Application Problem Graphs in Matrix Form.

$$XC = \begin{Bmatrix} 21 \\ 29 \\ 47 \\ 27 \\ 44 \\ 38 \\ 49 \end{Bmatrix} \qquad XC = \begin{Bmatrix} 27 \\ 35 \\ 24 \\ 33 \\ 48 \\ 38 \\ 41 \end{Bmatrix} \qquad XC = \begin{Bmatrix} 29 \\ 37 \\ 43 \\ 29 \\ 44 \\ 46 \\ 49 \end{Bmatrix}$$

Phase 1 **Phase 2** **Phase 3**

Figure 4.11 : Execution Cost of The Application Modules

phase are presented. A similar procedure can be used for the other phases.

Walk Through the Algorithm for the Example:

Step 0 : Initialize Procedure

Initialize procedure is called first to read P, XC, and S of phase 1. Next, it assignes heaps to every system link. Since we have 4 bidirectional links we need 8 heaps numbered 1,2,.., 8. The link to heap assignment matrix, EM which is created by Initialize is shown in Figure 4.12.

$$EM = \begin{vmatrix} 0 & 1 & 2 & 0 \\ 3 & 0 & 0 & 4 \\ 5 & 0 & 0 & 6 \\ 0 & 7 & 8 & 0 \end{vmatrix}$$

Figure 4.12 : Link-to-Heap Mapping Matrix.

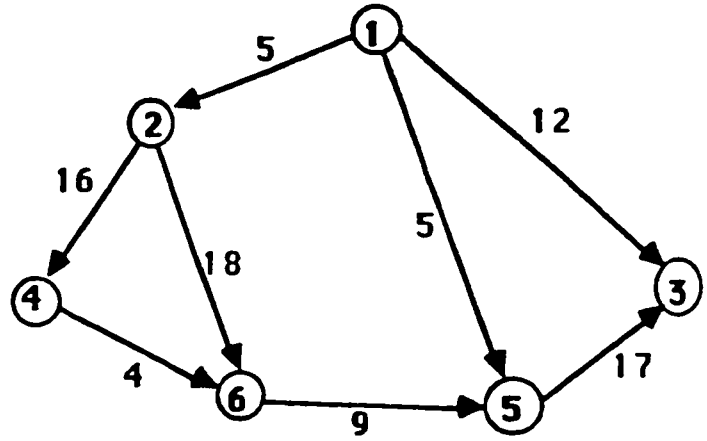
Step 1 : Clustering Procedure

Since n=4, clustering is called four times to reduce P into 1 cluster, 2 clusters, 3 clusters, and 4 clusters to be mapped onto 1 processor, 2 processors, 3 processors, and 4 processors respectively. The reason behind this is that we don't know if utilizing all processors or subset of them will be suboptimal. The steps of clustering modules into two clusters are given. Initially there exist 7 clusters containing one module each: Cluster 1 contains module m_1, cluster 2 contains m_2, ,....., and so on. The reduced problem matrix (PS) is initialized to be a copy of P. Next, the cd of every cluster pair is computed and pairs are stored in E-list and sorted according to their cd value. The sorted E-list will be as shown in Figure 4.13.

Cluster Pair		CD
i	j	
1	3	-8
2	5	-11
2	4	-11
1	2	-16
6	7	-21
4	5	-23
5	6	-29
3	6	-33
3	7	-35

Figure 4.13 :Content of E-list Before Graph Reduction Starts.

The pointer to every pair in E-list are stored in matrix edge. Since, pair (1,3) has the maximum cd, modules 1 and 3 are put in one cluster,1. Cluster 7 is renamed as cluster 3 because cluster 3 is empty. Formally, the array Cluster will be containing [1 2 1 4 5 6 3] as shown in Figure 4.14. After this clustering PD is updated. Also, pair (1,3) is removed from E-list. The resulting reduced graph represetned by PS will be as shown in Figure 4.14 (Note that the numbers inside the circles in this figure are cluster numbers).



(a)

Module	Cluster[Module]
1	1
2	2
3	1
4	4
5	5
6	6
7	3

(b)

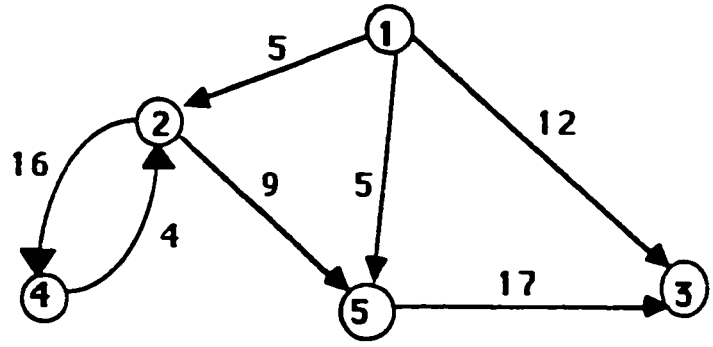
Figure 4.14 : The Reduced Graph and Cluster Matrix after Iteration 1.

Now the pairs that contain cluster 7 in E-list should be renamed as cluster 3. After that, the cd of pairs (1,2), (1,6) and (1,3) should be removed and reinserted again because the execution cost of cluster 1 is increased to 68 units. The resulting E-list is shown in Figure 4.15.

Cluster Pair		CD
i	j	
2	5	-11
2	4	-11
3	6	-21
4	5	-23
1	2	-24
5	6	-29
1	6	-33
1	3	-33

Figure 4.15 : E-list After the first clustering iteration.

The pair (2,5) is at the top of E-list. So cluster 2 and 5 should be joined. The resulting reduced graph is shown in Figure 4.16. Following the same procedure above, we



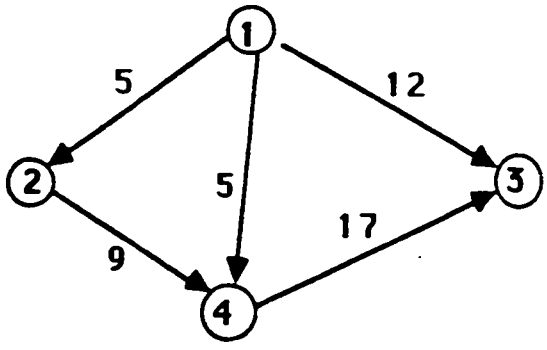
(a)

Module	Cluster[Module]
1	1
2	2
3	1
4	4
5	2
6	5
7	3

(b)

Figure 4.16 : The Reduced Graph and Cluster Matrix after Iteration 2.

continue reducing the problem graph until the number of clustering iterations is equal to 5; since m = 7 and we want to reduce the graph into two clusters, the number of iterations = 7-2 = 5. The reduced graph after iteration 3,4 and 5 is shown in Figure 4.17, 4.18 and 4.19 respectively.

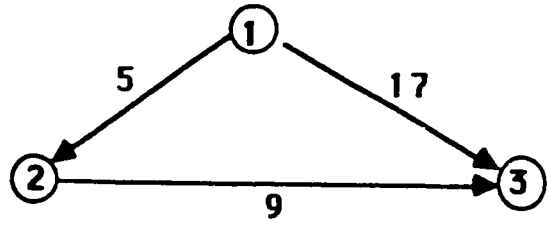


(a)

Module	Cluster[Module]
1	1
2	2
3	1
4	2
5	2
6	4
7	3

(b)

Figure 4.17 : The Reduced Graph and Cluster Matrix after Iteration 3.

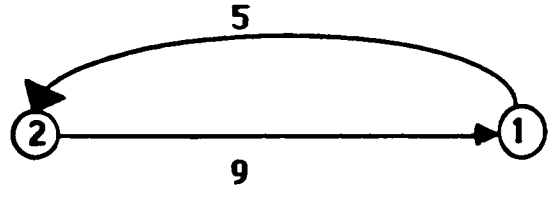


(a)

Module	Cluster[Module]
1	1
2	2
3	1
4	2
5	2
6	3
7	3

(b)

Figure 4.18 : The Reduced Graph and Cluster Matrix after Iteration 4.



(a)

Module	Cluster[Module]
1	1
2	2
3	1
4	2
5	2
6	1
7	1

(b)

Figure 4.19 : The Reduced Graph and Cluster Matrix after Iteration 5.

To reduce the graph into 3 clusters, the same procedure is followed. However, the number of iteration is less by one. The resulting reduced graph is the same as the one in Figure 4.18. Similarly, the reduced graph when we cluster into 4, is the same as the one in Figure 4.17.

Step 2 : Initial Assignment

For each one of the reduced graphs that resulted from step 1, we call Initial-assignment. We pass the reduced graph as a parameter. Since the operations applied on the four reduced graphs are the same, only obtaining the initial assignment for the reduced graph that obtains 4 clusters is presented. The matrix representation of this graph (PS) is shown in Figure 4.20. We will use the same graph in pairwise exchange. However, the resulting mapping of the other graphs will be given together with the OF.

Initially, CI, DEGP and DEGS matrices are calculated as explained in Section 4.2.3.1. The content of these matrices is shown in Figure 4.21. The first cluster selected for assignment is 4 because it has the maximum CI. It is mapped onto any system node because all of them have the same degree; say it is mapped onto 1. The next candidate cluster for assignment will be cluster 3 because it is a neighbor to

$$PS = \begin{Bmatrix} 0 & 5 & 12 & 5 \\ 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 17 & 0 \end{Bmatrix}$$

Figure 4.20 : The Resulting Reduced Graph with 4 Clusters).

$$CI = \begin{Bmatrix} 23 \\ 14 \\ 30 \\ 31 \end{Bmatrix} \qquad DEGP = \begin{Bmatrix} 3 \\ 2 \\ 2 \\ 3 \end{Bmatrix} \qquad DEGS = \begin{Bmatrix} 2 \\ 2 \\ 2 \\ 2 \end{Bmatrix}$$

Figure 4.21 : The Communication Intensity matrix (CI),The Degree Matrix of the Problem Graph (DEGP), and The Degree Matrix of the System Graph (DEGS).

cluster 4 and it has the largest CI. It can be mapped onto system nodes 2 or 3 because the maximum of the nominal distances between them and node 4 is 1. Let us choose node 2. We can't map cluster 3 onto system node 4 because the maximum nominal distance is 2. Continuing by this way, the initial mapping will be PSMAP = [3 4 2 1]. Note that the indices of PSMAP represent cluster numbers and entries represent processors to which clusters are assigned to.

Step 3 : Pairwise Exchange

The communication cost, OFC of this mapping is calculated by OF-calc. OFC of this mapping happen to be 29 and the candidate for exchange is cluster 1. Calling pairwise exchange procedure will find that exchange cluster 3 and 4 will decrease OFC to 18. So, the resulting mapping is [4 3 2 1]. Doing pairwise exchange again will not decrease OFC. So, we stop and the final mapping is returned as [4 3 2 1]. The total cost for this mapping is equal to 118 because the execution cost of this mapping is 100. The table in Figure 4.22 shows the mappings of clusters of the four reduced graphs, module to cluster mappings which define clusters, and the corresponding total costs. Since the minimum total cost among the four mappings is 117 which corresponds to the

# of clusters	Cluster-to-Processor Mapping *	Module-to-Cluster Mapping **	Total Cost	# of Pair-wise Exchanges
1	[1 2 3 4]	[1 1 1 1 1 1 1]	255	0
2	[1 2 3 4]	[1 2 1 2 2 1 1]	164	1
3	[3 4 1 2]	[1 2 1 2 2 3 3]	117	3
4	[4 3 2 1]	[1 2 1 2 2 4 3]	118	2

* Numbers within brackets are processor id's.

** Numbers within brackets are cluster id's.

Figure 4.22 : The Four Mappings of phase 1 .

reduced graphs with 3 clusters, we select the mapping of modules to processors of phase 1 as [3 1 3 1 1 2 2], See Figure 4.23.

Applying the same steps above to the problem graph of phase 2 and 3, the resulting suboptimal mappings are shown in Figure 4.24 and Figure 4.25 respectively. The application modules should be assigned as shown in Figure 4.23. After phase 1 completes modules m_1, m_2, m_3 and m_7 should be relocated because the suboptimal mapping for phase 2 is different for these modules. Similarly, modules m_3, m_4, m_6 and m_7 should be relocated in phase 3.

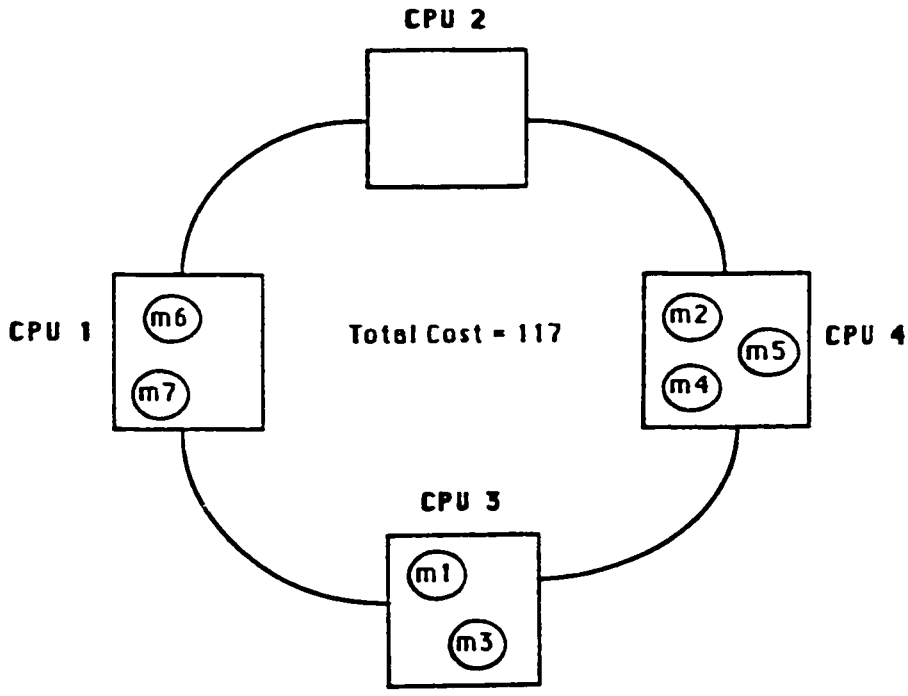


Figure 4.23 : Module-to-Processor Mapping at Phase 1

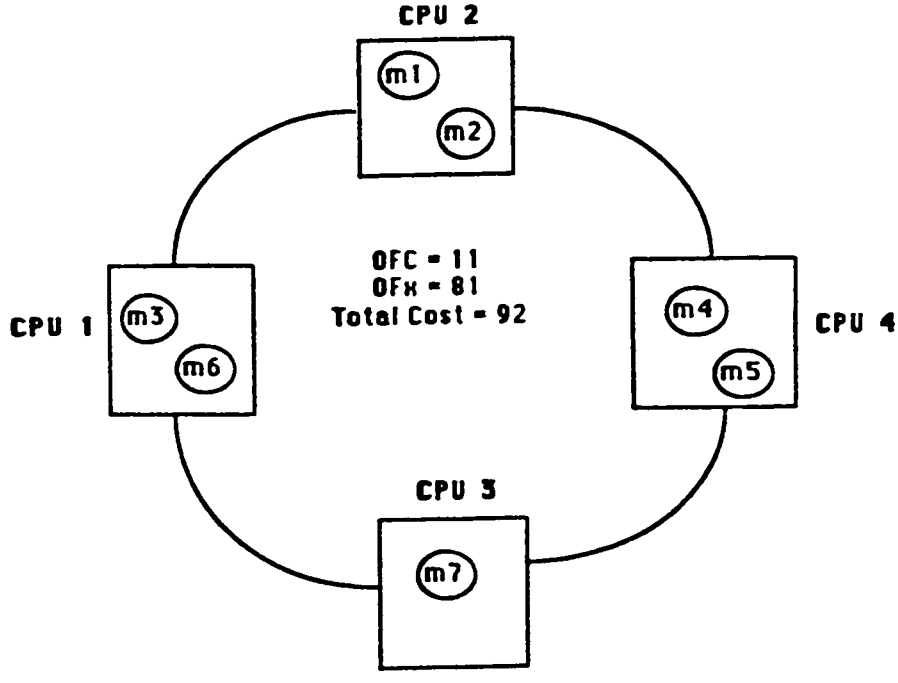


Figure 4.24 : Module-to-Processor Mapping at Phase 2

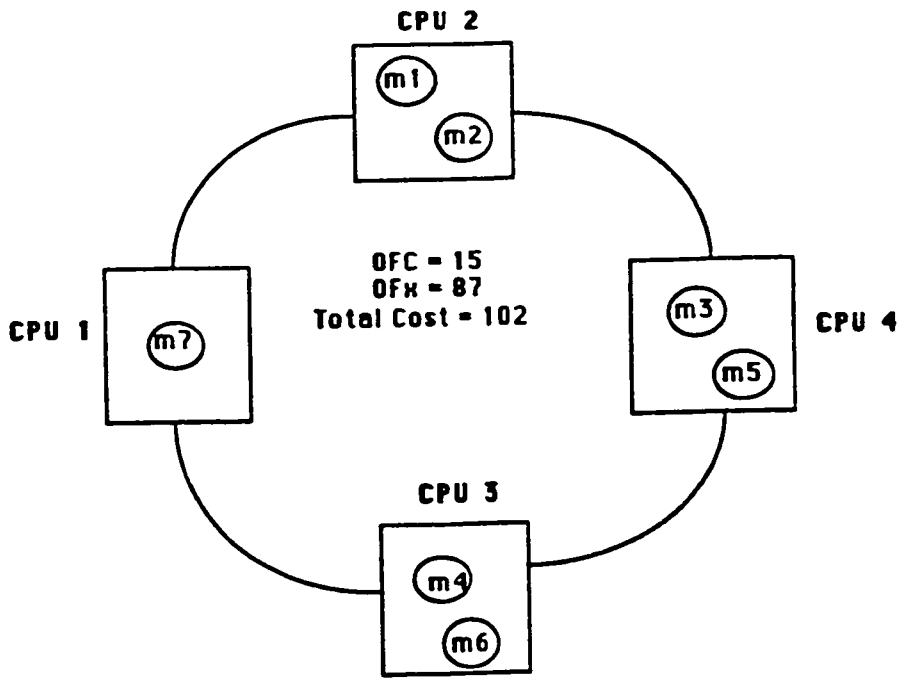


Figure 4.25 : Module-to-Processor Mapping at Phase 3

CHAPTER 5
PERFORMANCE ANALYSIS

5.1 Introduction

In order to evaluate the performance of the Load Balancing Algorithm (LBA), described in Chapter 3, 313 typical problem instances (test cases) were generated randomly. Each problem instance (PI for short) represents an application requirement at a phase whose modules are to be mapped onto a DCS modeled by a system graph. The number of modules, m, of the tested PI's ranges from 3 to 8, and the number of processors, n, ranges from 3 to 6. Appendix A contains a list of the PI's. As shown in the appendix, each PI is represented by a system graph (nxn matrix), a problem graph (mxm matrix) and an execution table of size m. In order to evaluate the quality of solutions obtained by the LBA, a branch and bound algorithm (BBA) has also been implemented[RICH83]. The implementation issues of the BBA are discussed in Section 5.2. In section 5.3, the results of both algorithms are shown and compared. The sections 5.4 and 5.5 present two modifications to the LBA. The first modification tries to improve the time complexity of the

LBA; and the new algorithm is called Bisection Based Load Balancing. The second modification attempts to improve the objective function of the algorithm while increasing slightly the time complexity. The performance of both algorithms is discussed. They show higher efficiency, compared to original algorithm.

5.2 A Branch and Bound Algorithm

In this section, an approach for finding optimal module-to-processor mappings of a given PI is presented. The problem is formulated as a state space search problem, and the well known A* algorithm is implemented to solve the problem.

All possible solutions to a problem define its solution space. The solution space of our problem consists of all possible module-to-processor mappings. Traditionally, a search space is diagramed as a tree with the initial state at the top and a state is connected to its successor by lines. A possible search space tree for a problem with 3 processors and 3 modules is shown in Figure 5.1. The root of the tree represents the initial state with no mapping. A pair (i,j) associated with a node in this tree represents assigning module i to processor j. A path in the tree from

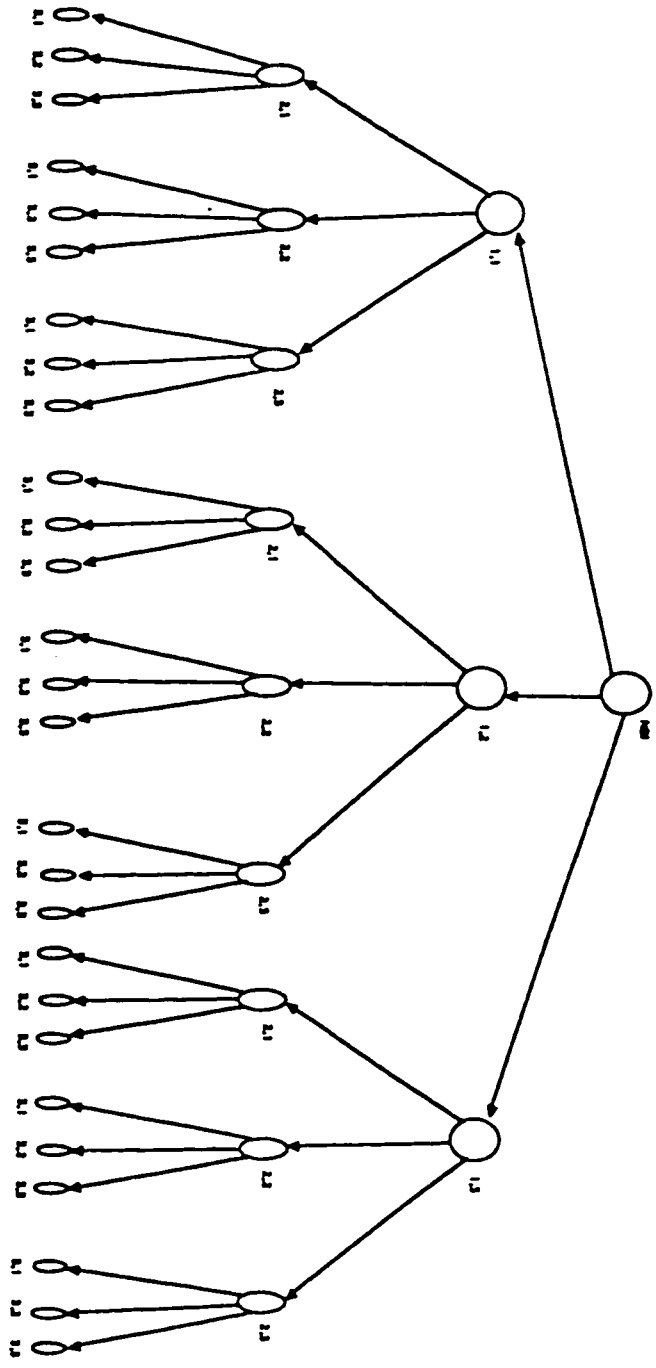


Figure 5.1 : A Search Space Tree for a problem with 5 processors and 3 modules.

the root node to a leaf defines one of the possible mappings. Hence, the solution space is the set of possible mappings that correspond to the set of all paths from the root to all leaf nodes.

The A* algorithm starts initially with the initial state and expands this node generating other nodes which in turn generate other nodes until the optimal solution is found. For some system and application graphs, the optimal solution may be found without expanding the tree to its upper limit; i.e. its solution space. Let us define some terminology before describing how the A* algorithm works. A node which has been generated and all of whose children are to be generated is called a live node. The live node whose children are being generated is called the E-node. In other words, the E-node is the node being expanded. Moreover, a dead node is a generated node that is either not to be expanded further or one for which all of its children has been generated. The set of live nodes are stored in a linked list called the Open list. For each path from the root to a node in level q, we associate a vector called MAPV (MAPing Vector) in the form $\{P_1,P_2,\ldots,P_q,0,0,\ldots,0\}$. This vector represents the mapping that corresponds to the path. For example the vector $\{P_1,P_2,\ldots,P_q,0,0,\ldots,0\}$ shows that

module 1 is mapped onto p1 , module 2 is mapped onto p2, ..., and module q is mapped onto P_q Modules i(i > 1) are not mapped yet. A variable U is defined as an upper bound on the cost of a minimum cost solution.

Initially, the A* algorithm starts with the root node as the only live node, and U is initialized to ∞. This node becomes the E-node and its MAPV is [0,0,...,0]. Next, we generate the n children of the current E-node. For each child node three functions are evaluated: Cost(MAPV), Lower Cost(MAPV), and Upper Cost(MAPV) where MAPV is the vector that corresponds to the child node. The three functions are defined as follows:

Cost (MAPV) : The total communication and execution cost of the mapping represented by MAPV; modules which are not mapped are ignored.

Lower Cost(MAPV): A lower bound on the total cost obtainable at a node such that Lower Cost (MAPV) ≤ Cost (V), where V is any complete mapping vector such that V[i]=MAPV[i] for i ≤ q. Let MAPV be $\{P_1,P_2,...,P_q,0,0,...,0\}$. Formally, we define Lower Cost(MAPV) as

$$\text{Lower Cost(MAPV)} = \frac{\sum_{i>q}^{m} XC(i)}{m-q} + \frac{1}{k} \sum_{i\le q, j>q}^{m} (P(i,j)+P(j,i)).$$

where XC(i) is the execution cost of module i, and k is the number of links of the system graph.

Upper(MAPV): defines an upper bound on the total cost obtained at a leaf node such that Upper Cost(MAPV) ≥ Cost(V) where V is defined as above. In our implementation, we defined Upper Cost(MAPV) as the cost of any random assignment in which the mapping of the first q modules is as given by MAPV. Since we are taking the cost of a feasible mapping and since the cost of an optimal mapping should be at most equal to or possiblly smaller than that of a feasible mapping,this function defines a correct upper bound.

Clearly, any child node of the current E-node whose Lower Cost(MAPV) ≥ Upper bound U should be killed; otherwise, it is added to Open List and U is set to Min (Upper Cost(MAPV),U). The nodes in Open list are ordered according to the level of a node in the search space tree and also

according to the order of generation within a level. After processing the current E-node, the node at the front of Open List is removed and becomes the E-node. The above E-node expansion method is repeated until one of the following happens:

i) Children of E-node are leaf nodes;or

ii) the OPEN list is empty.

A node X is considered as a leaf node if the mapping that corresponds to the path from the root to X is a full mapping; i.e. all modules are mapped. Initially, the cost of the optimal solution is set to ∞. If case (i) is encountered, we check if cost(X) is less than the cost of the previous optimal solution found so far. If it is less, the mapping that corresponds to node X is taken as the current optimal solution found; otherwise node X is ignored. If case (ii) is encountered the algorithm is terminated and the current optimal solution is returned and it will be an optimal solution for the problem.

For A* algorithm to be efficient and practical, it has to complete in a reasonable time; otherwise applying such an algorithm is not practical. The computation time depends on the application and system graph (or the problem instance).

5.3 Efficiency of the LBA

The utility of the LBA depends upon two factors:

i) the time needed to find suboptimal solutions, and

ii) the ratio between the cost objective functions of the mapping produced by the LBA (OF_h) and the optimal mapping produced by the BBA (OF_o).

The LBA has been applied to all the 313 PI's, some of which are listed in Appendix A. Table 5.1 gives the suboptimal mappings, the computing time, and OF_h of the listed PI's. The first column in the table contains the identification number of a PI.

The BBA could not be used to find optimal solutions for all PI's; some PI's need several days of CPU time to find their optimal solutions. Only 101 of the PI's were completed by the BBA in finite time. The results obtained by the BBA are shown in Table 5.2 in the same format as Table 5.1. Comparing the results of the LBA with the corresponding results obtained by the BBA, the former algorithm was found to be successful in finding suboptimal mappings. Figure 5.2 summarizes the comparison between the two algorithms by showing the distribution of cases for the ratio OF_h/OF_o.

PI	Time 1/100 sec	OFh	Module-to-processor Mapping
1	99	62	[1, 3, 2]
2	137	54	[1, 2, 3]
3	127	53	[1, 2, 3]
4	170	78	[2, 1, 3, 2]
5	385	67	[3, 1, 4, 2]
6	384	67	[3, 1, 4, 2]
7	368	67	[4, 3, 2, 1]
8	418	67	[2, 1, 4, 3]
9	88	69	[1, 3, 1, 2]
10	230	44	[1, 4, 2, 3]
11	253	44	[1, 4, 2, 3]
12	209	44	[3, 2, 1, 4]
13	220	44	[1, 4, 3, 2]
14	236	97	[1, 1, 3, 2]
15	494	68	[1, 4, 3, 2]
16	539	68	[1, 4, 3, 2]
17	467	68	[3, 2, 4, 1]
18	500	68	[1, 4, 2, 3]
19	77	121	[1, 2, 1, 2, 2]
20	258	93	[1, 3, 4, 2, 2]
21	252	94	[2, 3, 4, 1, 1]
22	247	93	[3, 4, 2, 1, 1]
23	258	93	[3, 4, 2, 1, 1]
24	922	67	[4, 3, 5, 1, 2]
25	577	67	[3, 4, 5, 2, 1]
26	566	67	[3, 5, 4, 2, 1]
27	599	67	[3, 4, 5, 2, 1]
28	1137	67	[1, 2, 5, 4, 3]
29	572	67	[3, 2, 5, 4, 1]
30	83	57	[1, 1, 3, 2, 2]
31	307	54	[2, 2, 3, 4, 1]
32	264	60	[2, 2, 4, 3, 1]
33	286	54	[1, 1, 4, 2, 3]
34	395	59	[3, 3, 2, 4, 1]
35	922	58	[5, 4, 2, 3, 1]
36	939	57	[4, 2, 3, 5, 1]
37	593	52	[1, 2, 5, 3, 4]
38	631	54	[1, 1, 4, 3, 2]
39	522	53	[5, 4, 2, 3, 1]
40	676	54	[1, 1, 4, 2, 3]
41	109	69	[3, 3, 2, 1, 2]
42	220	69	[3, 3, 1, 2, 1]
43	225	69	[3, 3, 2, 1, 2]
44	209	69	[4, 4, 3, 1, 3]
45	220	69	[2, 2, 3, 1, 3]
46	704	47	[4, 5, 3, 2, 1]
47	445	47	[3, 5, 4, 1, 2]
48	451	47	[5, 3, 4, 1, 2]
49	692	47	[3, 5, 4, 1, 2]
50	725	48	[3, 4, 2, 5, 1]
51	467	47	[3, 2, 5, 1, 4]

Table 5.1 The Results obtained by the LBA .

PI	Time 1/100 sec	OFh	Module-to-processor Mapping
52	77	108	[1, 1, 1, 2, 2]
53	193	76	[3, 2, 2, 4, 1]
54	236	77	[3, 1, 1, 4, 2]
55	192	76	[4, 1, 1, 2, 3]
56	412	76	[2, 3, 3, 1, 4]
57	824	57	[3, 2, 1, 5, 4]
58	599	57	[4, 1, 2, 5, 3]
59	445	57	[5, 1, 2, 4, 3]
60	698	57	[4, 1, 2, 5, 3]
61	621	57	[3, 4, 1, 5, 2]
62	818	57	[3, 5, 4, 1, 2]
63	138	140	[2, 3, 1, 1, 1]
64	374	95	[3, 4, 1, 2, 2]
65	527	104	[3, 4, 1, 2, 2]
66	374	95	[4, 2, 3, 1, 1]
67	439	95	[2, 4, 1, 3, 3]
68	1697	64	[3, 5, 4, 2, 1]
69	2076	64	[5, 4, 2, 1, 3]
70	1527	64	[4, 1, 2, 5, 3]
71	857	71	[4, 5, 2, 3, 1]
72	742	64	[2, 5, 4, 3, 1]
73	1164	65	[2, 5, 3, 4, 1]
74	159	125	[1, 2, 2, 2, 1, 3]
75	379	89	[1, 3, 3, 2, 1, 4]
76	412	90	[1, 3, 3, 2, 1, 4]
77	379	89	[3, 4, 4, 1, 3, 2]
78	423	89	[1, 2, 2, 3, 1, 4]
79	934	90	[1, 3, 3, 2, 1, 4]
80	944	89	[2, 5, 5, 1, 2, 4]
81	1065	89	[3, 2, 2, 1, 3, 5]
82	884	89	[2, 4, 4, 1, 2, 3]
83	835	89	[1, 4, 4, 2, 1, 3]
84	753	89	[4, 2, 2, 1, 4, 3]
85	1208	64	[1, 2, 5, 6, 3, 4]
89	116	136	[3, 2, 1, 1, 1, 3]
90	329	98	[4, 3, 2, 1, 1, 4]
104	121	119	[2, 3, 1, 1, 1, 3]
119	105	129	[2, 1, 3, 1, 1, 1]
134	93	140	[3, 1, 1, 1, 2, 1]
149	203	137	[2, 1, 3, 2, 1, 1, 1]
164	121	102	[3, 1, 2, 3, 1, 2, 1]
179	82	112	[1, 3, 3, 1, 1, 2, 2]
194	154	170	[3, 1, 1, 1, 2, 1, 3]
209	110	125	[3, 3, 2, 1, 3, 1, 1]
224	104	153	[1, 3, 2, 1, 1, 1, 3]
239	94	148	[3, 1, 3, 2, 1, 3, 3]
254	138	163	[2, 1, 1, 1, 2, 3, 3, 1]
269	88	127	[2, 1, 1, 1, 2, 1, 2, 2]
284	120	139	[2, 2, 1, 2, 1, 1, 2, 3]
299	138	136	[2, 3, 2, 3, 1, 1, 3, 1]

Table 5.1 The Results obtained by the LBA (Contd.).

PI	Time 1/100 sec	OFo	Module-to-processor Mapping
1	324	62	[1, 1, 2]
2	423	54	[1, 2, 3]
3	395	53	[1, 2, 3]
4	2394	78	[1, 2, 3, 1]
5	8700	67	[1, 2, 3, 4]
6	9596	67	[2, 1, 3, 4]
7	8014	67	[1, 2, 4, 3]
8	8590	67	[1, 2, 4, 3]
9	1395	65	[1, 2, 2, 3]
10	5180	44	[1, 2, 3, 4]
11	5657	44	[1, 2, 3, 4]
12	4817	44	[1, 2, 3, 4]
13	5185	44	[1, 4, 3, 2]
14	2773	92	[1, 2, 3, 2]
15	9920	68	[1, 2, 3, 4]
16	10694	68	[1, 2, 3, 4]
17	9244	68	[1, 2, 4, 3]
18	9882	68	[1, 4, 2, 3]
19	8410	82	[1, 2, 2, 3, 1]
20	40299	80	[1, 2, 4, 3, 2]
21	44808	80	[2, 3, 4, 1, 3]
22	36218	80	[1, 2, 3, 4, 2]
23	39882	80	[1, 2, 3, 4, 2]
24	15365	67	[2, 5, 3, 1, 4]
25	57915	67	[1, 3, 2, 5, 4]
26	61392	67	[1, 3, 2, 5, 4]
27	7900	67	[1, 5, 4, 2, 3]
28	57848	67	[1, 2, 3, 4, 5]
29	892	67	[1, 3, 2, 4, 5]
30	5124	57	[1, 1, 2, 3, 3]
31	26452	54	[2, 2, 3, 4, 1]
32	30517	57	[2, 1, 3, 4, 1]
33	22888	54	[1, 1, 2, 3, 4]
34	25699	54	[1, 1, 4, 3, 2]
35	35670	57	[2, 1, 3, 4, 1]
36	15775	52	[3, 2, 4, 1, 5]
37	18961	52	[1, 2, 3, 4, 5]
38	27986	54	[1, 1, 4, 3, 2]
39	16874	53	[1, 4, 3, 2, 5]
40	22998	52	[5, 4, 3, 1, 2]
41	5360	66	[1, 2, 2, 1, 3]
42	26277	59	[1, 2, 1, 3, 4]
43	29023	59	[1, 2, 1, 3, 4]
44	23728	59	[1, 2, 1, 3, 4]
45	25975	59	[1, 2, 1, 3, 4]
46	31095	47	[4, 5, 2, 3, 1]
47	16095	47	[1, 2, 3, 4, 5]
48	18627	47	[1, 2, 3, 4, 5]
49	26327	47	[1, 4, 5, 2, 3]
50	16139	47	[1, 2, 3, 4, 5]
51	22038	47	[1, 2, 3, 5, 4]

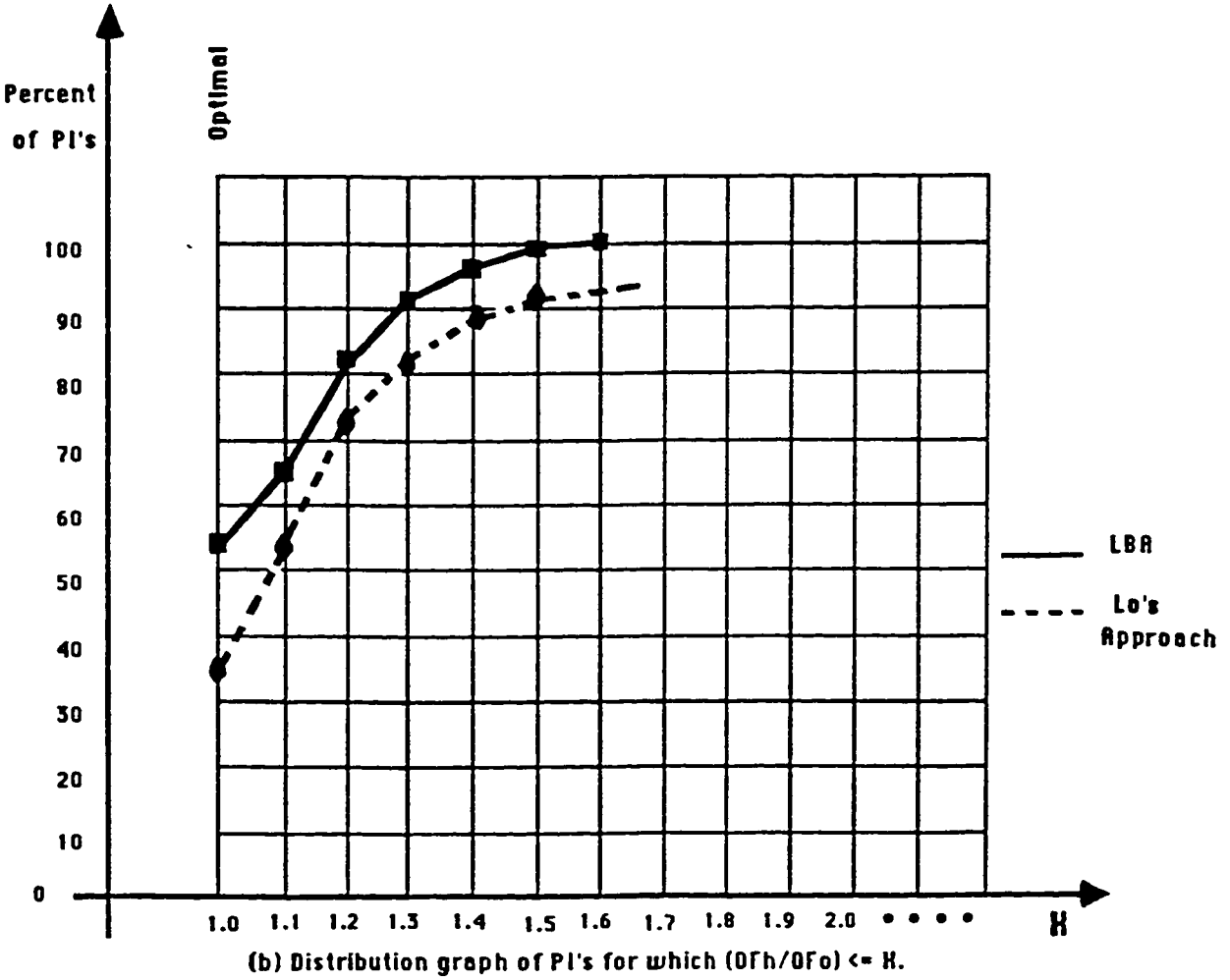
Table 5.2 : Results obtained by the Branch & Bound algorithm.

PI	Time 1/100 sec	OFo	Module-to-processor Mapping
52	6147	72	[1, 1, 2, 3, 3]
53	30407	61	[2, 1, 3, 1, 4]
54	33965	62	[2, 3, 1, 3, 4]
55	27051	61	[1, 2, 3, 2, 4]
56	29753	61	[1, 3, 2, 3, 4]
57	45452	57	[2, 3, 1, 4, 5]
58	27360	57	[1, 2, 3, 4, 5]
59	30249	57	[1, 2, 3, 4, 5]
60	39344	57	[1, 2, 4, 3, 5]
61	27909	57	[1, 2, 4, 3, 5]
62	34302	57	[1, 2, 3, 4, 5]
63	10276	98	[1, 1, 2, 2, 3]
64	51998	84	[4, 2, 3, 2, 1]
65	59353	97	[2, 1, 3, 1, 4]
66	45599	84	[1, 2, 3, 2, 4]
67	50543	85	[1, 2, 3, 2, 4]
68	61123	64	[2, 5, 4, 3, 1]
69	26822	64	[1, 2, 3, 4, 5]
70	31733	64	[1, 3, 5, 4, 2]
71	47271	64	[1, 5, 3, 4, 2]
72	28426	64	[1, 3, 4, 5, 2]
73	38802	64	[1, 2, 3, 5, 4]
74	44946	100	[1, 2, 3, 2, 1, 3]
75	300000	84	[1, 2, 3, 1, 4, 3]
76	350000	89	[1, 2, 2, 1, 3, 4]
77	260000	84	[3, 1, 4, 3, 2, 4]
78	290000	85	[1, 2, 4, 4, 3, 1]
79	140000	70	[1, 4, 1, 5, 2, 3]
80	1129235	70	[2, 1, 2, 5, 3, 4]
81	1140182	70	[2, 3, 2, 4, 1, 5]
82	1269251	70	[2, 3, 2, 5, 1, 4]
83	1116047	70	[4, 1, 4, 2, 3, 5]
84	1200331	70	[4, 1, 4, 2, 3, 5]
85	3410341	64	[1, 2, 5, 6, 3, 4]
89	44000	100	[1, 1, 2, 3, 2, 3]
90	290000	93	[1, 1, 2, 2, 3, 4]
104	29000	86	[1, 1, 2, 3, 2, 3]
119	27000	89	[1, 2, 3, 2, 3, 1]
134	31000	88	[1, 2, 3, 1, 3, 2]
149	120000	112	[1, 2, 3, 2, 1, 3, 3]
164	120000	102	[1, 2, 3, 1, 2, 3, 2]
179	100000	92	[1, 2, 3, 3, 3, 2, 1]
194	140000	119	[1, 1, 2, 1, 2, 3, 3]
209	120000	109	[1, 1, 2, 3, 1, 2, 3]
224	140000	115	[1, 1, 2, 2, 1, 3, 3]
239	130000	106	[1, 1, 2, 3, 1, 2, 3]
254	390000	119	[1, 2, 2, 3, 3, 3, 2, 1]
269	310000	95	[1, 2, 3, 3, 1, 3, 1, 2]
284	380000	103	[1, 2, 1, 3, 2, 2, 1, 3]
299	490000	132	[1, 2, 2, 3, 1, 1, 2, 3]

Table 5.2 : Results obtained by the Branch & Bound algorithm(Contd.)

K	Percent of cases for which (OFh/OFo) <= K	Total number of cases for which (K-1.0) < (OFh/OFo) <= K
1.0	52.5	53
1.1	66.4	14
1.2	80.3	14
1.3	90.2	10
1.4	97.1	7
1.5	99	2
1.6	100	1

(a) Distribution table.



(b) Distribution graph of PI's for which (OFh/OFo) <= K.

Figure 5.2 : Comparing the Performance of the LBR and the BBR.

For all of the 101 PI's, the LBA found an optimal mapping in 52.5 percent of the PI's. In 99 percent, the cost of the mapping produced by the LBA was found to be less than 1.5 times of OF_o. No ratios greater than 1.6 were found; only one instance produces a ratio of 1.6 and it was the worst. To put the results of our heuristic into perspective with results obtained by other heuristic approaches developed for the module assignment problem, the dotted line in Figure 5.2 shows the distribution of the same ratio obtained by LO[LO88]. She applied her algorithm for 536 cases. However, she formulated the problem differently by assuming that communication cost to be independent of the location of the communicating modules and independent of the communication interference of other modules. Note that a considerable amount of the time needed by our algorithm is for evaluating communication cost between modules.

5.4 Improving the Computation Time of the LBA

Recall that the LBA reduces the problem graph into n reduced graphs $G_i, i=1,2,\ldots,n$; where the size of G_i is i. Each G_i is mapped onto the system graph using the Initial-assignment and Pairwise-Exchange procedures, and

we choose the mapping with the minimum OF_q among the n mappings of the n Reduced graphs. Instead of reducing the problem graph into n different reduced graphs, we propose a method that is similar to the bisection method for decreasing the number of graphs, reduced and mapped, from n to $\log_2 n$; let us call this approach Bisection based LBA. Thus the overall complexity of the algorithm is reduced to $O(me\lg n\log e+n^2\lg n+N_i D_M a_M \lg n)$. Instead of reducing the problem graph into all possible sizes (i.e 1 through n), obtaining suboptimal mapping for every reduced graph, and selecting the mapping with the least cost, another strategy is followed. Selected sizes are taken instead of n. Most of problem instances behave as shown in Figure 5.3. The figure shows that the total cost (OF) of mapping modules decreases as the number of processors utilized for mapping is increased up to a certain minimum value after which it increases. But this is not true in all cases. Suppose that the minimum OF value is at processor p (see Figure 5.3). Note that for every number of processors utilized, the mapping should be optimal. The purpose of the Bisection Based LBA is to find this minimum. An outline of the algorithm is shown in Figure 5.4.

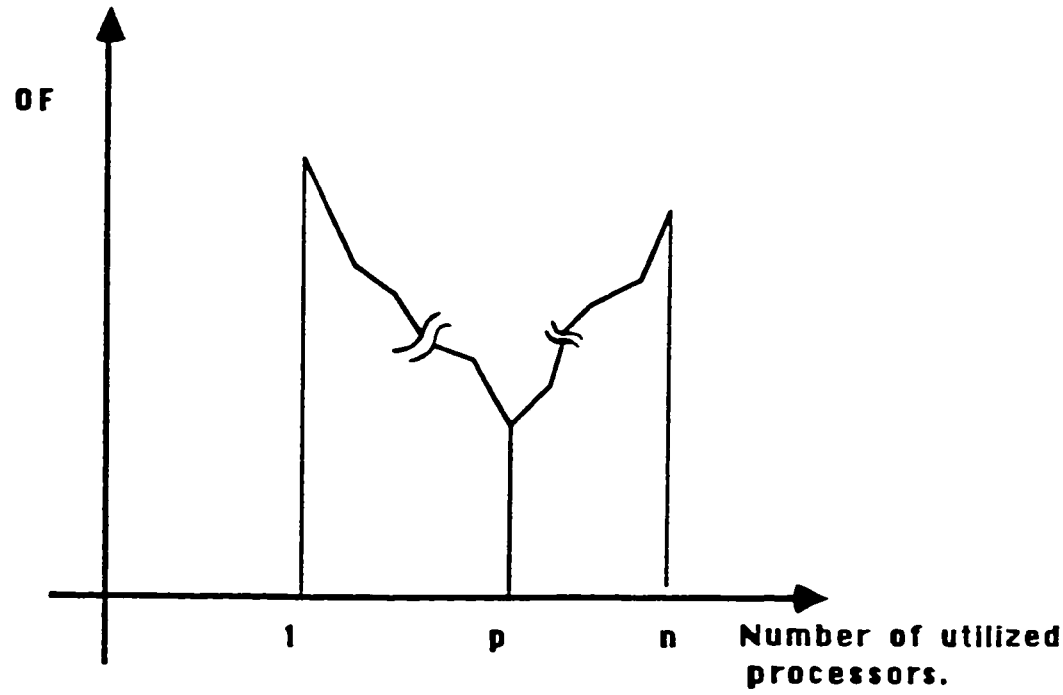


Figure 5.3 : A relation between the OF and the number of utilized processors.

```
Low = 1;High = n
For i = 1 to N Do
    OFA[i] = 0
  Repeat
    mid1 = (Low + high)/2
    mid2 = mid1 + 1
    IF ((mid1 = low) AND (mid1 = high)) Then
         Return mapping that corresponds to size q
         and exit.
    Else
         If OFA[mid1] = 0 Then
            Reduce P into a graph of size mid1,
            obtain suboptimal mapping and calculate
            its cost (OF1)
            OFA[mid1] = OF1
         Else
            OF1 = OFA[mid1]
         endif
         IF OFA[mid2] = 0 then
            Reduce P into a graph of size mid2,
            Obtain suboptimal mapping and calculate
            its cost (OF2); OFA[mid2] = OF2
         Else
            OF2 = OFA[mid2]
         Endif
         IF ((mid1 = Low) AND (mid1 <> high)) then
               IF OF1 < OF2 then
                    q = mid1
               Else
                    q = mid2
               Endif
               Return mapping that corresponds to size q
               and exit
          ELSE
               IF OF1 <= OF2 Then
                   high = mid1
               Else
                    low  = mid2
               Endif
         Endif
    Endif
  Until (True)
End.
```

Figure 5.4 : Outline of Bisection Based LBA.

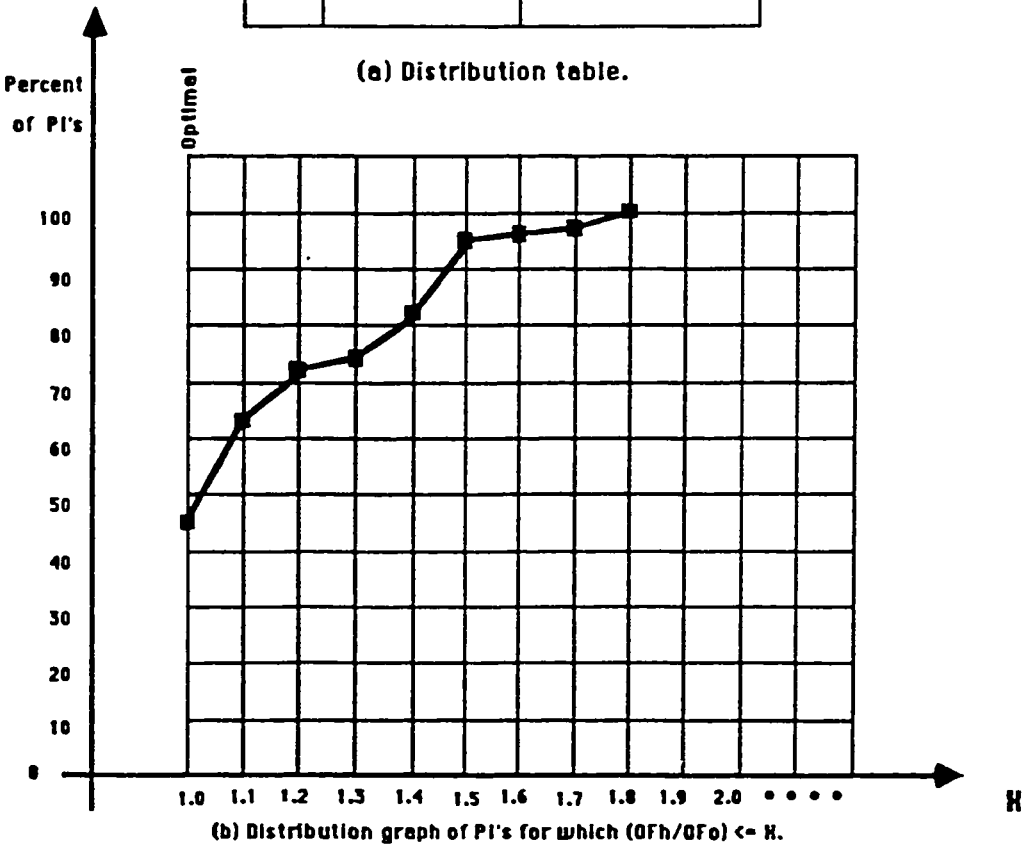
The Bisection based LBA has been applied to the PI's to which the BBA has been applied. Although the computing time of the algorithm was reduced by approximately 20% for the PI's, the suboptimal solutions obtained have been good. In addition, the algorithm has been successful to find the true minimum (see Figure 5.3) in 89% of the times. Figure 5.5 shows the distribution of the ratio OF_h/OF_o where OF_h is the OF for the Bisection based LBA.

5.5 LBA with Module Reassignment

If our aim of Load balancing was to optimize the results of the LBA, then it would have been a good idea to further improve the suboptimal solutions obtained by the LBA which are listed in Table 5.1. Starting with the solution obtained by the basic LBA, the optimization strategy we propose is to select a candidate cluster whose modules are to be allocated temporarily, one at a time, to the neighbor processors. The module which results in minimizing the old OF is reassigned to the corresponding location. This replacement method is repeated until the OF is not minimized. The candidate cluster should not be selected randomly. Instead, the cluster should be selected according to a certain criterion. For every cluster we evaluate the

K	Percent of cases for which (OFh/OFo) <= K	Total number of cases for which (K-1.0) < (OFh/OFo) <= K
1.0	46.8	44
1.1	62.7	15
1.2	70.1	7
1.3	75.4	5
1.4	81.8	6
1.5	94.6	12
1.6	95.7	1
1.7	96.8	1
1.8	100	3

(a) Distribution table.



(b) Distribution graph of PI's for which (OFh/OFo) <= K.

Figure 5.5 : Performance of the Bisection Based LBR..

sum of execution cost and the maximum communication cost among the edges to which the cluster connects. The cluster that has the maximum sum is selected as the candidate. With this modification, the module-to-processor mapping of all the 313 PI's would be as shown in Table 5.3. The last column presents the number of iterations in module replacements. As we see from the table, the number of iterations is not a large number for all the 313 PI's and most of them require no more than one trial. Figure 5.6 shows the OF_h/OF_o ratio distribution with module replacement. The number of optimal mappings found by the new algorithm is 71.3 percent which was 52.5 for the basic LBA. Also, the worst ratio was found to be 1.4 whereas it was 1.6 by the basic LBA. Depending on the importance of turn-around time of the application one can apply the basic LBA, bisection based LBA, or LBA with module replacement. If the turn-around is very critical, the best algorithm to use is the LBA with module replacement. On the other hand, if turn-around time is not critical, then one of the other algorithms may be applied.

PI	Time 1/100 sec	OFh	Module-to-Processor Mapping	Number of Iterations in Module Replacement
1	76	62	[2 2 1]	1
2	110	54	[1 2 3]	1
3	99	53	[1 2 3]	1
4	1191	78	[2 1 3 2]	1
5	451	67	[3 1 4 2]	1
6	396	67	[3 1 4 2]	1
7	445	67	[4 3 2 1]	1
8	423	67	[2 1 4 3]	1
9	203	65	[1 3 3 2]	2
10	329	44	[1 4 2 3]	1
11	303	44	[1 4 2 3]	1
12	341	44	[3 2 1 4]	1
13	308	44	[1 4 3 2]	1
14	434	92	[1 2 3 2]	2
15	631	68	[1 4 3 2]	1
16	599	68	[1 4 3 2]	1
17	637	68	[3 2 4 1]	1
18	505	68	[1 4 2 3]	1
19	154	117	[1 1 1 2 2]	2
20	379	80	[1 3 4 2 3]	2
21	483	80	[2 3 4 1 3]	2
22	351	80	[3 4 2 1 4]	2
23	373	80	[3 4 2 1 4]	2
24	808	67	[4 3 5 1 2]	1
25	615	67	[3 4 5 2 1]	1
26	588	67	[3 5 4 2 1]	1
27	594	67	[3 4 5 2 1]	1
28	1060	67	[1 2 5 4 3]	1
29	593	67	[3 2 5 4 1]	1
30	186	57	[1 1 3 2 2]	1
31	379	54	[2 2 3 4 1]	1
32	324	57	[2 1 4 3 1]	2
33	374	54	[1 1 4 2 3]	1
34	467	59	[3 3 2 4 1]	1
35	758	57	[2 1 4 3 1]	2
36	863	57	[4 2 3 5 1]	1
37	593	52	[1 2 5 3 4]	1
38	708	54	[1 1 4 3 2]	1
39	516	53	[5 4 2 3 1]	1
40	736	54	[1 1 4 2 3]	1
41	231	67	[1 3 2 1 2]	2
42	472	66	[2 3 4 2 1]	2
43	336	67	[1 3 2 1 2]	2
44	505	59	[2 4 2 1 3]	2
45	417	59	[4 2 4 3 1]	2
46	582	47	[4 5 3 2 1]	1
47	462	47	[3 5 4 1 2]	1
48	659	47	[5 3 4 1 2]	1
49	775	47	[3 5 4 1 2]	1
50	939	48	[3 4 2 5 1]	1
51	560	47	[3 2 5 1 4]	1

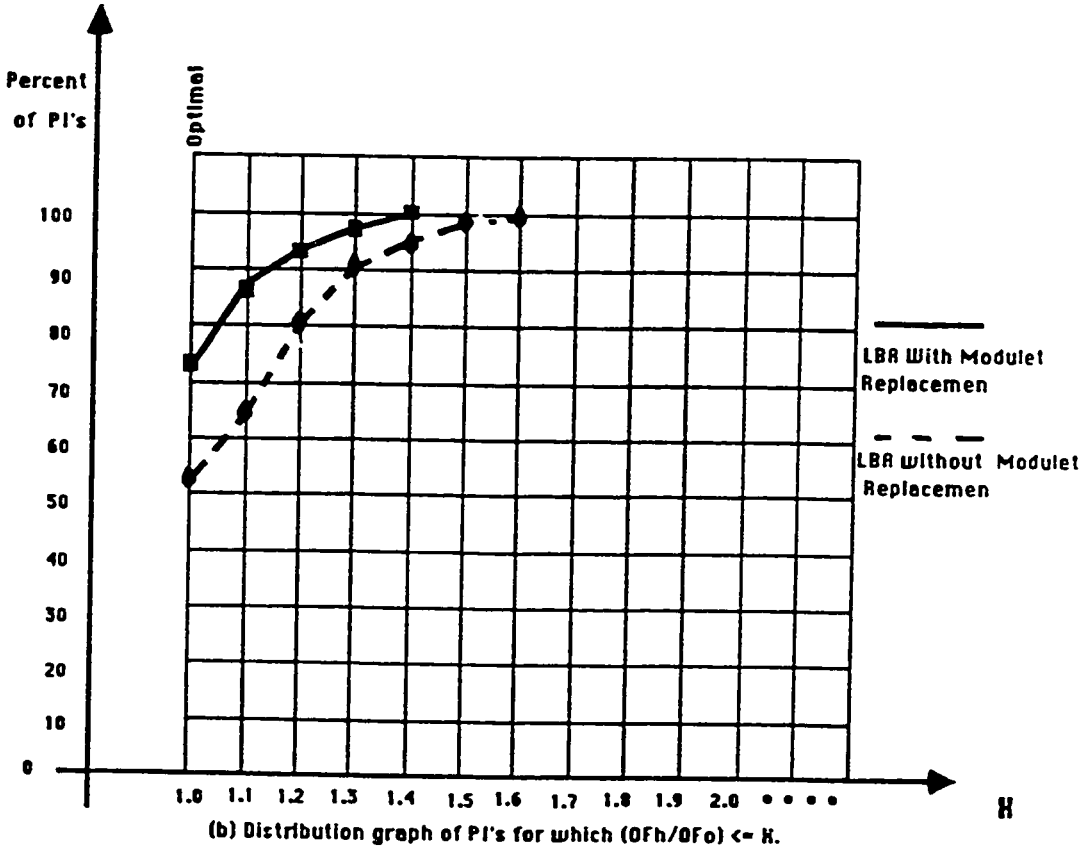
Table 5.3 : The results of LBA with module reassignment.

PI	Time 1/100 sec	OFh	Module-to-Processor Mapping	Number of Iterations in Module Replacement
52	335	73	[3 1 2 3 1]	3
53	560	70	[3 3 2 4 1]	2
54	362	77	[3 1 1 4 2]	1
55	676	64	[4 2 1 2 3]	3
56	521	73	[1 1 3 4 4]	4
57	972	57	[3 2 1 5 4]	1
58	989	57	[4 1 2 5 3]	1
59	907	57	[5 1 2 4 3]	1
60	889	57	[4 1 2 5 3]	1
61	934	57	[3 4 1 5 2]	1
62	988	57	[3 5 4 1 2]	1
63	615	98	[2 3 3 1 1]	2
64	824	94	[3 4 1 1 2]	2
65	939	98	[2 3 3 1 1]	2
66	912	94	[4 2 3 3 1]	2
67	1028	95	[2 4 1 3 3]	1
68	1730	64	[3 5 4 2 1]	1
69	2180	64	[5 4 2 1 3]	1
70	1620	64	[4 1 2 5 3]	1
71	1258	71	[4 5 2 3 1]	1
72	1252	64	[2 5 4 3 1]	1
73	1577	65	[2 5 3 4 1]	1
74	418	100	[1 2 3 2 1 3]	2
75	687	89	[1 3 3 2 1 4]	1
76	741	90	[1 3 3 2 1 4]	1
77	747	89	[3 4 4 1 3 2]	1
78	802	89	[1 2 2 3 1 4]	1
79	1505	90	[1 3 3 2 1 4]	1
80	1538	89	[2 5 5 1 2 4]	1
81	1614	89	[3 2 2 1 3 5]	1
82	1884	86	[1 3 5 4 2 1]	2
83	1378	89	[1 4 4 2 1 3]	1
84	1807	85	[3 1 5 2 4 3]	2
85	1659	64	[1 2 5 6 3 4]	1
89	488	109	[3 2 1 1 2 3]	2
90	775	98	[4 3 2 1 1 4]	1
104	395	91	[2 3 1 1 2 3]	2
119	296	128	[2 1 2 2 1 1]	3
134	318	127	[1 2 1 1 2 2]	3
149	626	114	[2 3 3 2 1 1 1]	2
164	357	102	[3 1 2 3 1 2 1]	1
179	418	94	[1 3 3 3 1 2 2]	2
194	1055	126	[3 1 1 3 2 2 3]	3
209	593	109	[3 3 2 1 3 2 1]	2
224	686	121	[1 3 2 2 1 1 3]	2
239	1010	105	[1 1 3 2 1 2 3]	5
254	868	125	[2 1 1 3 2 3 3 1]	2
269	357	97	[2 3 1 3 2 3 1 2]	2
284	241	139	[2 2 1 2 1 1 2 3]	1
299	434	136	[2 3 2 3 1 1 3 1]	1

Table 5.3 : The results of LBA with module reassignment(Contd.).

K	Percent of cases for which (OFh/OFo) <= K	Total number of cases for which (K-1.0) < (OFh/OFo) <= K
1.0	71.3	72
1.1	88.1	17
1.2	92.1	4
1.3	97.0	5
1.4	100	3

(a) Distribution table.



(b) Distribution graph of PI's for which (OFh/OFo) <= K.

Figure 5.6 : Comparing Performance of the Basic LBR and LBR with Module Replacement.

CHAPTER 6
CONCLUSIONS AND SUGGESTIONS

In this thesis, we have addressed the problem of load balancing in assigning modules of distributed applications into distributed computing systems. A mathematical model has been developed. The model is based on the phase concept of a distributed application. Requirements of an application are assumed to change dynamically. The period of time, during which a number of modules execute in parallel and communicate with each other at stable communication and execution loads, is called a phase. The model developed here handles both the cases where number of modules is less than and greater than the number of processors.

Our investigation of the problem has resulted in the development of three heuristic algorithms. The first algorithm has three steps. First, the problem graph is reduced into a graph of size of the system graph or less. This approach allows reduced complexity for the mapping step. In a reduced graph, nodes are called clusters which contain one or more modules. Each reduced graph is mapped

using a step-wise refinement method. For example initial assignment gives a coarse mapping and pairwise exchange technique provides further balancing after the initial assignment. Test results indicate that the algorithm performs well on a variety of problem and system graphs. Altogether 313 problem and system graphs are used to test the algorithm. The results of the algorithm have been compared with optimal solutions obtained by a branch and bound algorithm which could be applied to only 101 of problem and system graphs. For about 52% of the total cases, the solutions produced by the algorithm were optimal, and the worst case was found to have a cost of 1.6 times the cost of the optimal solution.

The second algorithm is the same as the first one except that we reduce the problem graph into selective sizes using a bisection method. The percentage of cases for which the solution obtained by the second algorithm were optimal is 47.4, and the worst case had a cost of 1.8 times the cost of the optimal solution. The third algorithm improves the assignment of modules by reforming clusters. This algorithm was the best one; for 71.3% of the total number of cases, were optimal and the worst case happened to be 1.4 times the cost of the optimal solution.

The following are some of the further investigation points need to be carried out:

1) Consideration of module relocation cost:

When modules are reassigned from phase to phase, the cost of relocation can be significant. Given module relocation cost, an efficient algorithm is needed to minimize the sum of communication, execution and relocation costs.

2) Handling more than one application on the same system.

This problem is very important especially for general purpose distributed systems. In such systems hardware and software resources are shared. Changes in the load distribution are more dynamic and interaction causes higher complexity. The load balancing under these conditions, should include integration of loads from different applications.

APPENDIX A
LIST OF PROBLEM INSTANCES USED IN TESTING ALGORITHMS

Problem Instance # 1

$$P=\begin{Bmatrix}0 & 0 & 17\\ 7 & 0 & 0\\ 0 & 7 & 0\end{Bmatrix} \quad XC=\begin{Bmatrix}20\\ 20\\ 45\end{Bmatrix} \quad S=\begin{Bmatrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{Bmatrix}$$

Problem Instance # 2

$$P=\begin{Bmatrix}0 & 15 & 0\\ 0 & 0 & 11\\ 12 & 0 & 0\end{Bmatrix} \quad XC=\begin{Bmatrix}32\\ 39\\ 28\end{Bmatrix} \quad S=\begin{Bmatrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{Bmatrix}$$

Problem Instance # 3

$$P=\begin{Bmatrix}0 & 15 & 0\\ 0 & 0 & 9\\ 12 & 0 & 0\end{Bmatrix} \quad XC=\begin{Bmatrix}32\\ 38\\ 23\end{Bmatrix} \quad S=\begin{Bmatrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{Bmatrix}$$

Problem Instance # 4

$$P=\begin{Bmatrix}0 & 0 & 0 & 13\\ 10 & 0 & 19 & 0\\ 0 & 0 & 0 & 0\\ 0 & 14 & 0 & 0\end{Bmatrix} \quad XC=\begin{Bmatrix}26\\ 36\\ 48\\ 33\end{Bmatrix} \quad S=\begin{Bmatrix}0 & 1 & 1\\ 1 & 0 & 1\\ 1 & 1 & 0\end{Bmatrix}$$

Problem Instance # 5

$$P=\begin{Bmatrix}0 & 0 & 0 & 13\\ 10 & 0 & 19 & 0\\ 0 & 0 & 0 & 0\\ 0 & 14 & 0 & 0\end{Bmatrix} \quad XC=\begin{Bmatrix}26\\ 36\\ 48\\ 33\end{Bmatrix} \quad S=\begin{Bmatrix}0 & 1 & 1 & 1\\ 1 & 0 & 1 & 0\\ 1 & 1 & 0 & 0\\ 1 & 0 & 0 & 0\end{Bmatrix}$$

Problem Instance # 6

$$P=\begin{Bmatrix}0 & 0 & 0 & 13\\ 10 & 0 & 19 & 0\\ 0 & 0 & 0 & 0\\ 0 & 14 & 0 & 0\end{Bmatrix} \quad XC=\begin{Bmatrix}26\\ 36\\ 48\\ 33\end{Bmatrix} \quad S=\begin{Bmatrix}0 & 1 & 1 & 1\\ 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0\\ 1 & 0 & 0 & 0\end{Bmatrix}$$

Problem Instance # 7

$$P=\begin{Bmatrix} 0 & 0 & 0 & 13 \\ 10 & 0 & 19 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 14 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 36 \\ 48 \\ 33 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 8

$$P=\begin{Bmatrix} 0 & 0 & 0 & 13 \\ 10 & 0 & 19 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 14 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 36 \\ 48 \\ 33 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 9

$$P=\begin{Bmatrix} 0 & 10 & 0 & 11 \\ 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 32 \\ 27 \\ 26 \\ 29 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 10

$$P=\begin{Bmatrix} 0 & 10 & 0 & 11 \\ 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 32 \\ 27 \\ 26 \\ 29 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 11

$$P=\begin{Bmatrix} 0 & 10 & 0 & 11 \\ 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 32 \\ 27 \\ 26 \\ 29 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 12

$$P=\begin{Bmatrix} 0 & 10 & 0 & 11 \\ 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 32 \\ 27 \\ 26 \\ 29 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 13

$$P=\begin{Bmatrix} 0 & 10 & 0 & 11 \\ 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 \\ 0 & 0 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 32 \\ 27 \\ 26 \\ 29 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 14

$$P=\begin{Bmatrix} 0 & 11 & 17 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17 \\ 19 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 49 \\ 29 \\ 43 \\ 44 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 15

$$P=\begin{Bmatrix} 0 & 11 & 17 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17 \\ 19 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 49 \\ 29 \\ 43 \\ 44 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 16

$$P=\begin{Bmatrix} 0 & 11 & 17 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17 \\ 19 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 49 \\ 29 \\ 43 \\ 44 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 17

$$P=\begin{Bmatrix} 0 & 11 & 17 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17 \\ 19 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 49 \\ 29 \\ 43 \\ 44 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 18

$$P=\begin{Bmatrix} 0 & 11 & 17 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 17 \\ 19 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 49 \\ 29 \\ 43 \\ 44 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 19

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 20

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 21

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 22

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 23

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 24

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 25

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 26

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 27

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 28

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 29

$$P=\begin{Bmatrix} 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 12 \\ 0 & 0 & 0 & 0 & 0 \\ 12 & 11 & 0 & 0 & 0 \\ 0 & 0 & 0 & 19 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 29 \\ 34 \\ 48 \\ 32 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 30

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 8 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 7 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 24 \\ 20 \\ 42 \\ 27 \\ 22 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 31

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 8 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 7 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 24 \\ 20 \\ 42 \\ 27 \\ 22 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 32

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 8 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 7 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 24 \\ 20 \\ 42 \\ 27 \\ 22 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 33

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 8 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 7 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 24 \\ 20 \\ 42 \\ 27 \\ 22 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 34

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 8 & 0 & 8 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 & 0 \\ 0 & 0 & 7 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 24 \\ 20 \\ 42 \\ 27 \\ 22 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 35

$$P=\begin{bmatrix}0&0&0&0&0\\9&0&8&0&8\\0&0&0&0&0\\0&7&0&0&0\\0&0&7&10&0\end{bmatrix}\quad XC=\begin{bmatrix}24\\20\\42\\27\\22\end{bmatrix}\quad S=\begin{bmatrix}0&1&1&1&0\\1&0&0&0&0\\1&0&0&0&0\\1&0&0&0&1\\0&0&0&1&0\end{bmatrix}$$

Problem Instance # 36

$$P=\begin{bmatrix}0&0&0&0&0\\9&0&8&0&8\\0&0&0&0&0\\0&7&0&0&0\\0&0&7&10&0\end{bmatrix}\quad XC=\begin{bmatrix}24\\20\\42\\27\\22\end{bmatrix}\quad S=\begin{bmatrix}0&1&0&0&1\\1&0&1&1&0\\0&1&0&0&1\\0&1&0&0&1\\1&0&1&1&0\end{bmatrix}$$

Problem Instance # 37

$$P=\begin{bmatrix}0&0&0&0&0\\9&0&8&0&8\\0&0&0&0&0\\0&7&0&0&0\\0&0&7&10&0\end{bmatrix}\quad XC=\begin{bmatrix}24\\20\\42\\27\\22\end{bmatrix}\quad S=\begin{bmatrix}0&1&0&0&0\\1&0&1&0&1\\0&1&0&1&1\\0&0&1&0&1\\0&1&1&1&0\end{bmatrix}$$

Problem Instance # 38

$$P=\begin{bmatrix}0&0&0&0&0\\9&0&8&0&8\\0&0&0&0&0\\0&7&0&0&0\\0&0&7&10&0\end{bmatrix}\quad XC=\begin{bmatrix}24\\20\\42\\27\\22\end{bmatrix}\quad S=\begin{bmatrix}0&1&0&1&0\\1&0&1&1&0\\0&1&0&0&1\\1&1&0&0&0\\0&0&1&0&0\end{bmatrix}$$

Problem Instance # 39

$$P=\begin{bmatrix}0&0&0&0&0\\9&0&8&0&8\\0&0&0&0&0\\0&7&0&0&0\\0&0&7&10&0\end{bmatrix}\quad XC=\begin{bmatrix}24\\20\\42\\27\\22\end{bmatrix}\quad S=\begin{bmatrix}0&1&0&1&0\\1&0&0&1&0\\0&0&0&1&1\\1&1&1&0&1\\0&0&1&1&0\end{bmatrix}$$

Problem Instance # 40

$$P=\begin{bmatrix}0&0&0&0&0\\9&0&8&0&8\\0&0&0&0&0\\0&7&0&0&0\\0&0&7&10&0\end{bmatrix}\quad XC=\begin{bmatrix}24\\20\\42\\27\\22\end{bmatrix}\quad S=\begin{bmatrix}0&1&0&1&0\\1&0&1&0&0\\0&1&0&1&0\\1&0&1&0&1\\0&0&0&1&0\end{bmatrix}$$

Problem Instance # 41

$$P=\begin{bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{bmatrix}\quad XC=\begin{bmatrix}23\\33\\20\\30\\34\end{bmatrix}\quad S=\begin{bmatrix}0&1&1\\1&0&1\\1&1&0\end{bmatrix}$$

Problem Instance # 42

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&1&1\\1&0&1&0\\1&1&0&0\\1&0&0&0\end{Bmatrix}$$

Problem Instance # 43

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&1&1\\1&0&0&0\\1&0&0&0\\1&0&0&0\end{Bmatrix}$$

Problem Instance # 44

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&0&1&1\\0&0&1&1\\1&1&0&1\\1&1&1&0\end{Bmatrix}$$

Problem Instance # 45

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&0&1&1\\0&0&1&1\\1&1&0&0\\1&1&0&0\end{Bmatrix}$$

Problem Instance # 46

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&1&1&0\\1&0&0&0&0\\1&0&0&0&0\\1&0&0&0&1\\0&0&0&1&0\end{Bmatrix}$$

Problem Instance # 47

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&0&0&1\\1&0&1&1&0\\0&1&0&0&1\\0&1&0&0&1\\1&0&1&1&0\end{Bmatrix}$$

Problem Instance # 48

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&0&0&0\\1&0&1&0&1\\0&1&0&1&1\\0&0&1&0&1\\0&1&1&1&0\end{Bmatrix}$$

Problem Instance # 49

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&0&1&0\\1&0&1&1&0\\0&1&0&0&1\\1&1&0&0&0\\0&0&1&0&0\end{Bmatrix}$$

Problem Instance # 50

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&0&1&0\\1&0&0&1&0\\0&0&0&1&1\\1&1&1&0&1\\0&0&1&1&0\end{Bmatrix}$$

Problem Instance # 51

$$P=\begin{Bmatrix}0&13&0&0&0\\0&0&0&11&0\\0&0&0&0&0\\0&0&0&0&13\\9&0&7&0&0\end{Bmatrix}\quad XC=\begin{Bmatrix}23\\33\\20\\30\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&0&1&0\\1&0&1&0&0\\0&1&0&1&0\\1&0&1&0&1\\0&0&0&1&0\end{Bmatrix}$$

Problem Instance # 52

$$P=\begin{Bmatrix}0&5&0&0&6\\0&0&16&0&0\\12&0&0&0&7\\0&0&0&0&0\\0&3&0&8&0\end{Bmatrix}\quad XC=\begin{Bmatrix}31\\23\\41\\22\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&1\\1&0&1\\1&1&0\end{Bmatrix}$$

Problem Instance # 53

$$P=\begin{Bmatrix}0&5&0&0&6\\0&0&16&0&0\\12&0&0&0&7\\0&0&0&0&0\\0&3&0&8&0\end{Bmatrix}\quad XC=\begin{Bmatrix}31\\23\\41\\22\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&1&1\\1&0&1&0\\1&1&0&0\\1&0&0&0\end{Bmatrix}$$

Problem Instance # 54

$$P=\begin{Bmatrix}0&5&0&0&6\\0&0&16&0&0\\12&0&0&0&7\\0&0&0&0&0\\0&3&0&8&0\end{Bmatrix}\quad XC=\begin{Bmatrix}31\\23\\41\\22\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&1&1&1\\1&0&0&0\\1&0&0&0\\1&0&0&0\end{Bmatrix}$$

Problem Instance # 55

$$P=\begin{Bmatrix}0&5&0&0&6\\0&0&16&0&0\\12&0&0&0&7\\0&0&0&0&0\\0&3&0&8&0\end{Bmatrix}\quad XC=\begin{Bmatrix}31\\23\\41\\22\\34\end{Bmatrix}\quad S=\begin{Bmatrix}0&0&1&1\\0&0&1&1\\1&1&0&1\\1&1&1&0\end{Bmatrix}$$

Problem Instance # 56

$$P=\begin{Bmatrix} 0 & 5 & 0 & 0 & 6 \\ 0 & 0 & 16 & 0 & 0 \\ 12 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 23 \\ 41 \\ 22 \\ 34 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 57

$$P=\begin{Bmatrix} 0 & 5 & 0 & 0 & 6 \\ 0 & 0 & 16 & 0 & 0 \\ 12 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 23 \\ 41 \\ 22 \\ 34 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 58

$$P=\begin{Bmatrix} 0 & 5 & 0 & 0 & 6 \\ 0 & 0 & 16 & 0 & 0 \\ 12 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 23 \\ 41 \\ 22 \\ 34 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 59

$$P=\begin{Bmatrix} 0 & 5 & 0 & 0 & 6 \\ 0 & 0 & 16 & 0 & 0 \\ 12 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 23 \\ 41 \\ 22 \\ 34 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 60

$$P=\begin{Bmatrix} 0 & 5 & 0 & 0 & 6 \\ 0 & 0 & 16 & 0 & 0 \\ 12 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 23 \\ 41 \\ 22 \\ 34 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 61

$$P=\begin{Bmatrix} 0 & 5 & 0 & 0 & 6 \\ 0 & 0 & 16 & 0 & 0 \\ 12 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 23 \\ 41 \\ 22 \\ 34 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 62

$$P=\begin{Bmatrix} 0 & 5 & 0 & 0 & 6 \\ 0 & 0 & 16 & 0 & 0 \\ 12 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 8 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 23 \\ 41 \\ 22 \\ 34 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 63

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 64

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 65

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 66

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 67

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 68

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 69

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 70

$$P=\begin{pmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{pmatrix} \quad XC=\begin{pmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{pmatrix} \quad S=\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Problem Instance # 71

$$P=\begin{pmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{pmatrix} \quad XC=\begin{pmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{pmatrix} \quad S=\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Problem Instance # 72

$$P=\begin{pmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{pmatrix} \quad XC=\begin{pmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{pmatrix} \quad S=\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Problem Instance # 73

$$P=\begin{pmatrix} 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 13 & 0 & 0 & 18 \\ 0 & 0 & 9 & 0 & 0 \\ 18 & 0 & 0 & 12 & 0 \end{pmatrix} \quad XC=\begin{pmatrix} 45 \\ 35 \\ 45 \\ 31 \\ 46 \end{pmatrix} \quad S=\begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Problem Instance # 74

$$P=\begin{pmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad XC=\begin{pmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{pmatrix} \quad S=\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Problem Instance # 75

$$P=\begin{pmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad XC=\begin{pmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{pmatrix} \quad S=\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Problem Instance # 76

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 77

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 78

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 79

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 80

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 81

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 82

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 83

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 84

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 85

$$P=\begin{Bmatrix} 0 & 0 & 0 & 15 & 18 & 16 \\ 10 & 0 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 26 \\ 46 \\ 25 \\ 38 \\ 45 \\ 41 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 89

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 14 & 0 & 0 \\ 10 & 0 & 0 & 0 & 17 & 0 \\ 0 & 19 & 15 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 25 \\ 49 \\ 38 \\ 36 \\ 43 \\ 40 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 90

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 15 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 14 & 0 & 0 \\ 10 & 0 & 0 & 0 & 17 & 0 \\ 0 & 19 & 15 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 25 \\ 49 \\ 38 \\ 36 \\ 43 \\ 40 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{Bmatrix}$$

Problem Instance # 104

$$P=\begin{Bmatrix} 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 13 & 0 & 0 \\ 0 & 0 & 0 & 0 & 15 & 14 \\ 14 & 0 & 5 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 36 \\ 23 \\ 31 \\ 34 \\ 40 \\ 37 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 119

$$P=\begin{Bmatrix} 0 & 0 & 7 & 0 & 0 & 0 \\ 5 & 0 & 0 & 11 & 0 & 9 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 11 & 0 & 0 & 0 \\ 6 & 14 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 30 \\ 37 \\ 46 \\ 30 \\ 26 \\ 25 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 134

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 12 & 0 & 0 & 0 & 8 & 9 \\ 0 & 17 & 0 & 11 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 31 \\ 44 \\ 31 \\ 28 \\ 35 \\ 24 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 149

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 \\ 11 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 18 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 \\ 0 & 3 & 19 & 0 & 19 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 30 \\ 26 \\ 49 \\ 47 \\ 48 \\ 23 \\ 21 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 164

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 6 & 11 & 14 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 16 & 0 \\ 14 & 0 & 6 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 3 & 0 & 0 & 0 & 0 & 1 \\ 0 & 6 & 6 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 36 \\ 25 \\ 48 \\ 30 \\ 36 \\ 42 \\ 20 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 179

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 15 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 13 & 0 & 0 \\ 17 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 9 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 44 \\ 39 \\ 21 \\ 20 \\ 34 \\ 36 \\ 25 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 194

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 17 & 0 & 0 & 17 & 0 \\ 0 & 0 & 0 & 7 & 0 & 0 & 12 \\ 0 & 8 & 0 & 0 & 7 & 0 & 0 \\ 0 & 6 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 4 & 9 & 0 & 0 \\ 11 & 0 & 0 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 29 \\ 37 \\ 43 \\ 29 \\ 44 \\ 45 \\ 49 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 209

$$P=\begin{Bmatrix} 0 & 0 & 8 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 17 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 19 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 & 15 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 21 \\ 29 \\ 47 \\ 27 \\ 44 \\ 38 \\ 49 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 224

$$P=\begin{Bmatrix} 0 & 5 & 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 13 & 0 & 8 & 15 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 27 \\ 37 \\ 35 \\ 42 \\ 33 \\ 38 \\ 47 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 239

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 13 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 11 & 0 & 0 & 0 \\ 0 & 0 & 16 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 16 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 22 \\ 43 \\ 40 \\ 47 \\ 24 \\ 42 \\ 33 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 254

$$P=\begin{Bmatrix} 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 & 0 & 18 \\ 0 & 5 & 0 & 0 & 4 & 5 & 0 & 0 \\ 6 & 4 & 0 & 0 & 0 & 0 & 6 & 0 \\ 9 & 0 & 0 & 6 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 13 & 9 & 0 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 39 \\ 31 \\ 34 \\ 39 \\ 31 \\ 30 \\ 36 \\ 46 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 269

$$P=\begin{Bmatrix} 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 9 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 8 & 15 \\ 0 & 13 & 12 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9 & 0 & 0 \\ 5 & 0 & 0 & 0 & 0 & 0 & 6 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 24 \\ 35 \\ 31 \\ 25 \\ 20 \\ 23 \\ 36 \\ 38 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 284

$$P=\begin{Bmatrix} 0 & 10 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 13 & 8 & 0 \\ 0 & 0 & 4 & 0 & 0 & 0 & 0 & 11 \\ 19 & 0 & 0 & 12 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 \end{Bmatrix} \quad XC=\begin{Bmatrix} 48 \\ 26 \\ 21 \\ 32 \\ 25 \\ 33 \\ 22 \\ 47 \end{Bmatrix} \quad S=\begin{Bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{Bmatrix}$$

Problem Instance # 299

$$P=\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 18 & 0 & 0 & 0 & 0 \\ 16 & 0 & 0 & 0 & 0 & 3 & 7 & 0 \\ 0 & 0 & 0 & 0 & 5 & 5 & 7 & 0 \\ 0 & 0 & 6 & 0 & 0 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 18 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 0 & 8 & 0 & 0 & 0 \end{pmatrix} \qquad XC=\begin{pmatrix} 41 \\ 23 \\ 49 \\ 46 \\ 35 \\ 37 \\ 36 \\ 46 \end{pmatrix} \qquad S=\begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

REFERENCES

1. [AGR88] R. Agrawal, and H. Jagadish,"Partitioning techniques for large-grained parallelism,"IEEE TOC, Vol.37, No. 12, Dec. 1988.

2. [BARA85] A. BARAK, and A. Shiloh,"A distributed load balancing Policy for a multicomputer, "Software-Practice and Experience,Vol. 15(9),Sept. 1985.

3. [BOKH79] S. H. Bokhari,"Dual processor scheduling with dynamic reassignment,"IEEE Trans. on software Engineering(TOSE), vol.SE-5, No.4, July 1979.

4. [BOKH81] _____ ,"On the mapping problem,"IEEE Trans. on computers(TOC), vol. C-30, No. 3, March 1981.

5. [BOKH88] _____ ,"Partitioning problems in parallel, pipelined, and distributed computing,"IEEE TOC, vol. C-37, No.1, Jan 1988.

6. [BOZ89] M. Bozyigit,U. Kalaycioglu,and M. Melhi,"Load balancing in dense distributed systems,"Proceedings of The Fourth International Symposium on Computer and Information Sciences, Turkey, Vol. 1, October 1989,pp. 345-361.

7. [CHO82] T. CHON, and J. Abraham,"Load balancing in distributed systems ,"IEEE TOSE,Vol. SE-8,No.4, July 1982.

8. [CHU80] Wesly Chu, L. Holloway, M. Land, and Kemal Efe,"Task allocation in distributed data processing," IEEE Computer,Nov. 1980.

9. [EFE82] Kemal Efe ,"Heuristic models of task assignment scheduling in distributed systems," IEEE Computer,June 1982.

10. [LEE85] Soo-Young Lee and J. K. Aggarwal,"A mapping strategy for parallel processing," IEEE TOC, Vol.36, No.4, Dec. 1987.

11. [LIN87] Frank Lin ,and R. Keller,"The gradient model load balancing method ,"IEEE TOSE,Vol. SE-13,No.1 ,Jan. 1987.

12. [LO88] V. M. Lo,"Heuristic algorithms for task assignment in distributed systems," IEEE TOC,Vol. 37, No.11, Nov. 1988.

13. [PET85] J. Peterson, and A. Silberschatz,Operating Systems Concepts, Addison-Wesley, Canada, 2nd Ed.,1985.

14. [RICH83] E. Rich,Artificial Intelegence,McGraw-Hill,1983.

15. [STAN84] J. Stankovic ,"A perspective in distributed computer systems ,"IEEE TOC, Vol. C-33,No.12, Dec. 1984.

16. [STON77] H. S. Stone,"Multiprocessor Scheduling with the aid of network flow algorithms," IEEE TOSE,Vol. SE-3,No.1,Jan. 1977.

17. [STON78] H. S. Stone, and S. Bokhari,"Control of distributed processes,"IEEE Computer,July 1978.

18. [TAN85] A. Tanenbaum and R. Renesse,"Distributed operating systems," Computing surveys,Vol.17, No.4, Dec. 1985.